Universidad
Carlos III de Madrid

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

PROYECTO FIN DE CARRERA

Puesta en marcha del Sensor Inercial MTi de Xsens

Autor: Daniel Morales Tejero

Tutor: Santiago Martínez de la Casa Díaz

Titulación**:** I.T.I. Electrónica Industrial

Noviembre 2011

TITULO: Puesta en marcha del Sensor Inercial MTi de Xsens

AUTOR: Daniel Morales Tejero

TUTOR: Santiago Martínez de la Casa Díaz

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 14 de Noviembre de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

SECRETARIO VOCAL PRESIDENTE

RESUMEN

El departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid está desarrollando el robot humanoide TEO con fines de investigación en este campo. Una de las partes fundamentales es la estabilidad de dicho humanoide y para ello, se ha realizado este proyecto. El objetivo de éste es la integración del sensor inercial MTi de la marca Xsens, para que genere los datos adecuados de posición y orientación, que serán usados en el control del robot.

Para llevar a cabo este proyecto, se ha realizado una aplicación básica, escrita en C++, bajo la plataforma Linux, que sirve para configurar el dispositivo y obtener los datos.

También, se ha implementado una interfaz gráfica en Matlab. En dicha interfaz, el usuario podrá elegir la velocidad de transmisión del dispositivo y se representaran los datos de la aceleración, velocidad angular y campo magnético de los tres ejes (X, Y, Z). Además es posible capturar estos datos para luego representarlos en una serie de gráficas y poder analizarlos.

Por último, se ha desarrollado un módulo de comunicación para transmitir los datos capturados del sensor a la interfaz gráfica.

Una vez hecha toda la programación, se han realizado diversas pruebas para comprobar su funcionamiento.

ABSTRACT

The Systems Engineering and Automation Department of the University Carlos III of Madrid is developing the humanoid robot TEO with aims of investigation in this field. One fundamental part of this work is the stability of humanoid robot. It has motivated the development of this project. The objective of this one is the integration of the inertial sensor MTi from Xsens, so that it generates the data adapted of position and direction that will be used in the stability control of the robot.

To carry out this project, there has been developed a basic application, written in C++, under the Linux platform, whose objective is to configure the device and get data.

Also, it has been implemented a graphical user interface in Matlab. In this interface, the user will be able to choose the speed of transmission of the device and the data of the acceleration, angular velocity and magnetic field of the three axes (X, Y, Z). In addition, it is possible to capture these data to represent them in a series of graphs. Moreover, the movement of the sensor can be represented by the angles of Euler.

It also has been developed a communication module to transmit the data captured from the sensor to the graphical user interface.

Finally, it has been carried out various tests to verify their operation.

ÍNDICE

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1: INTRODUCCIÓN

1.1 INTRODUCCIÓN

La robótica es la ciencia encaminada a diseñar y construir aparatos y sistemas capaces de realizar o ayudar al ser humano a desempeñar tareas. Dentro de esta amplia ciencia existen distintas especialidades, como la robótica industrial, la de servicio, la inteligente o la humanoide, esta última es el la que está centrada nuestro proyecto.

La importancia de la robótica en nuestras vidas es evidente, hoy en día muchas funciones que anteriormente realizaban los seres humanos son realizadas robots. Desde las tareas más cotidianas, hasta los procesos industriales más complejos y peligrosos. Por lo que, ya que los robots realizan tareas propias de los humanos, se comenzaron a desarrollar robots humanoides. Estos robots han de ser capaces de adaptarse a distintos tipos de entornos, imitar funciones humanas como andar, subir pendientes, manipular objetos, interactuar con el exterior, etc.

Por ello en el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid se están realizando diversos proyectos e investigaciones en este ámbito. Uno de esos proyectos en desarrollo es el robot humanoide TEO, esta versión mejorada viene precedida por otras dos versiones anteriores pertenecientes a la serie RH. Consiste en un sistema mecánico de 26 (24+2 de la cabeza) grados de libertad, dispuestos de modo que le dotan de apariencia humana tanto en el aspecto como en la capacidad de locomoción.

1.2 ANTECEDENTES

El Proyecto de Robots Humanoides RH de la Universidad Carlos III de Madrid consta con un prototipo ya construido (RH-1) y uno en fase de desarrollo (TEO). En este apartado se pretende dar una visión general de los aspectos más importantes del diseño de estos robots [10] [11].

1.2.1 El RH-1.

En el año 2006 se creó el robot RH-1 en el Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid.

Este robot se definió como un sistema mecánico de 21 grados de libertad (GDL) todos ellos activos. En cada GDL hay un motor encargado de su movimiento y están dispuestos de modo que le dotan de apariencia humana tanto en el aspecto como en la capacidad de locomoción, con las limitaciones propias de la extrema complejidad del sistema locomotor de un ser humano. El RH-1 fue el resultado del rediseño de algunos elementos hardware de su predecesor RH-0 (conservando los grados de libertad). Ambos fueron desarrollados tomando como modelos los prototipos más avanzados existentes en aquel momento: el ASIMO de Honda y el HRP-2P de Kawada.

A continuación, en la Figura 1, se comparan los tres robots que sirvieron de modelo para la realización del robot RH-1.

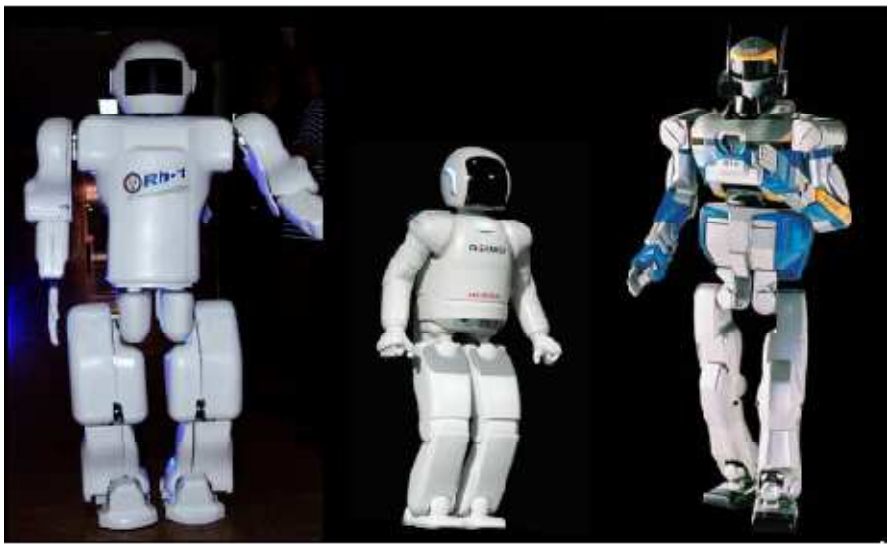

Figura 1: Comparativa de los robots RH1 (izqda.), ASIMO (centro) y HRP-2P (dcha.)

Se recurría a una arquitectura descentralizada, usando para ello dos redes según el protocolo CAN-bus. Una red para la parte superior, y otra para la parte inferior, dedicadas ambas al control de los drivers de los motores. La distribución incluye el uso de varios microprocesadores.

Es común a ambos sistemas hardware, el uso de un microprocesador de formato PC/104 con bus internos ISA dedicado al tratamiento digital de imagen y sonido, y otro igual para el control de la red CAN y el control de las trayectorias de las articulaciones en función de los datos sensoriales. Para determinar la posición angular de los motores, se obtenía la información directamente del encoder relativo incluido el motor.

La comunicación entre los microprocesadores y el router era inalámbrica. Para ello se incluía mediante conexión una tarjeta inalámbrica según el protocolo 802.11b. Para comunicarse con los inclinómetros y giroscopios se recurría a una conexión serie RS-232.

En la Figura 2 se muestra el resultado final del robot humanoide RH-1.

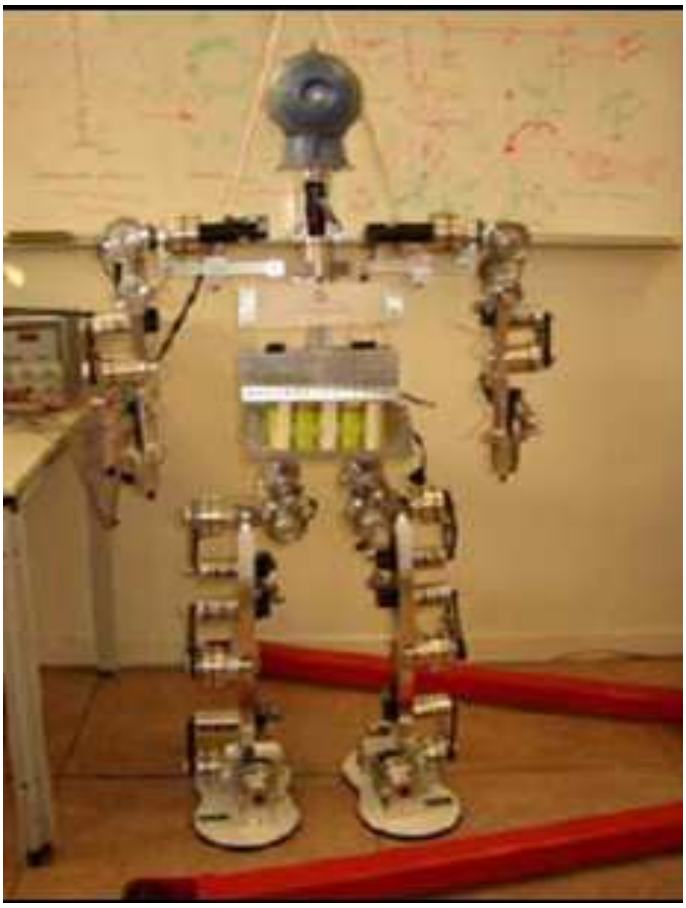

Figura 2: RH1

1.2.2 TEO

Para este nuevo robot que se está creando actualmente, se estima un peso de 55 Kg y una velocidad de 3 Km/h durante la caminata. Se calcula que podrá transportar objetos de 4 Kg de peso por cada brazo e incluso sentarse. Su altura aumenta de 1.2 m a 1.65 m, dotando al robot de un tamaño más acorde al de un humano. Las medidas del robot se pueden observar en la Figura 3, que se muestra a continuación.

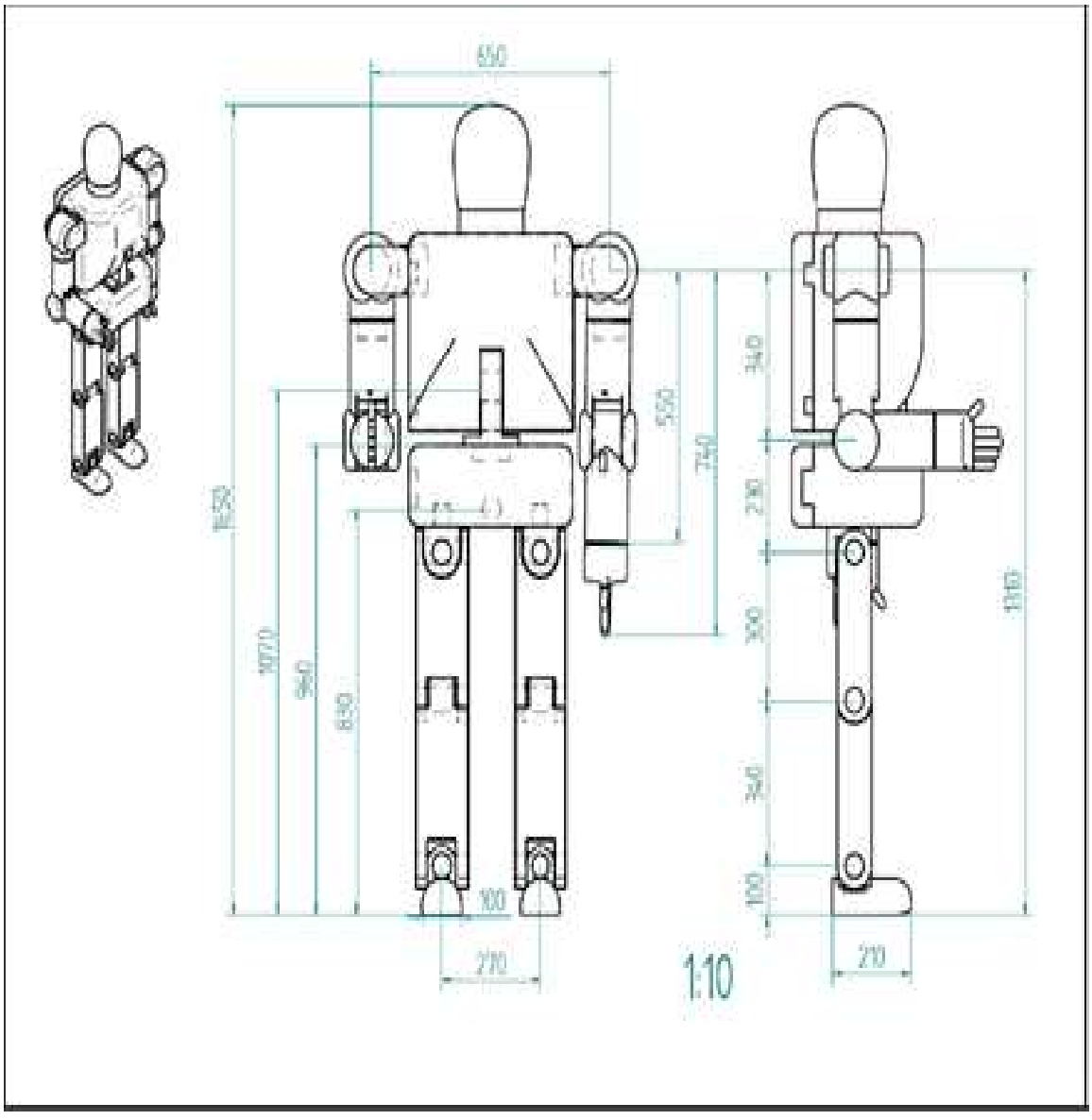

Figura 3: Medidas prototipo TEO

Para el diseño de este robot, se ha buscado una solución para los problemas derivados de las anteriores versiones y teniéndolos en cuenta, se ha diseñado un nuevo sistema actualizado acorde a las nuevas tecnologías disponibles. Los anteriores modelos disponían de 21 grados de libertad sin contar con la cabeza. El TEO pretende incluir tres grados más de libertad: uno en cada codo en el eje transversal que permita a los brazos realizar movimientos más parecidos a los humanos y otro grado de libertad en el tronco en su eje sagital para poder controlar de manera más rápida el balanceo del cuerpo hacia delante y atrás y lograr mantener su centro de masa centrado. En la Tabla 1 se detallan los grados de libertad en cada parte del cuerpo del robot.

Tabla 1: Grados de libertad TEO

GDL	Número	Eje de movimiento
PIERNAS	12 GDL	
Cadera	3 (x2)	Sagital Frontal Transversal
Rodilla	1 (x2)	Sagital
Tobillo	2 (x2)	Sagital Frontal
BRAZOS	10 GDL	
Hombros	2 (x2)	Sagital Frontal
Codo	2 (x2)	Sagital Transversal (nuevos)
Muñeca	1 (x2)	Transversal
TRONCO	2 GDL	
Tronco	2	Transversal Sagital (nuevo)
CABEZA	2 GDL	
Cámara	2	Trasversal (izquierda-derecha) Sagital(arriba-abajo)
TOTAL	26 GDL con cabeza 24 GDL sin cabeza	

1.3 OBJETIVOS Y ALCANCE DEL PROYECTO

En este proyecto queremos realizar una aplicación básica de comunicación, con la cual, poder integrar el sensor inercial MTi de la marca Xsens en el humanoide TEO, utilizando Linux como sistema operativo y el lenguaje de programación C++. Para ello se han propuesto los siguientes objetivos secundarios:

- Estudiar el sensor y las funciones que proporciona para implementar el componente software.
- Diseñar e implementar un módulo de comunicación básica con el sensor.
- Desarrollar una interfaz gráfica, en Matlab, que sirva para que el usuario pueda configurar el sensor, además, los datos se verán representados de forma más amigable para usuarios y desarrolladores, y no sólo en modo texto mediante una terminal.

1.4 ESTRUCTURA Y CONTENIDO DE LA MEMORIA

A continuación se detallará la estructura de este proyecto que consta de 5 capítulos.

- En este primer capítulo, se ha realizado una introducción explicando cuales han sido las motivaciones que han dado lugar a la realización del presente proyecto, proponiendo para su ejecución una serie de objetivos a cumplimentar en el desarrollo del mismo.

- En el segundo capítulo, se describen los elementos usados, es decir, se describen las características del sensor inercial MTi de la marca Xsens.

- El tercer capítulo explica la herramienta software utilizada, Matlab, en la interfaz del programa.

- En el cuarto capítulo, se desarrolla el trabajo, explicando cómo se comunica el sensor y el modo de salida de los datos, la manera que tiene de comunicarse el usuario a través de la interfaz gráfica con el MTi y el funcionamiento del programa realizado.

- El capítulo quinto muestra las pruebas realizadas con el sensor para verificar su buen funcionamiento.

- En el sexto capítulo, se exponen las conclusiones finales y los trabajos futuros que se pueden realizar con ésta aplicación.

- En el último capítulo, el séptimo, se detalla el presupuesto del proyecto.

Capítulo 2: HARDWARE UTILIZADO

2.1 SENSOR INERCIAL MTi DE XSENS

En esta sección en la que nos encontramos están basadas en las referencias [1] [2] y [9]. Para una mayor información consultar dichas referencias.

2.1.1 Visión general del sistema MTi

El MTi (‘*Motion Tracker*’) es un sensor de medición inercial con magnetómetros 3D integrados, con un procesador integrado capaz de calcular ‘*roll*’, ‘*pitch*’ y ‘*yaw*’ en tiempo real, así como, la salida calibrada 3D de aceleración lineal, velocidad angular (giroscopio) y los datos del campo magnético.

Nosotros utilizamos la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) correspondiente a la librería de C++ "XsensCMTstatic.Lib", proporcionada por el proveedor del sensor.

En la Figura 4 se muestran los módulos de los que está compuesto el sensor por dentro y sus conexiones entre módulos.

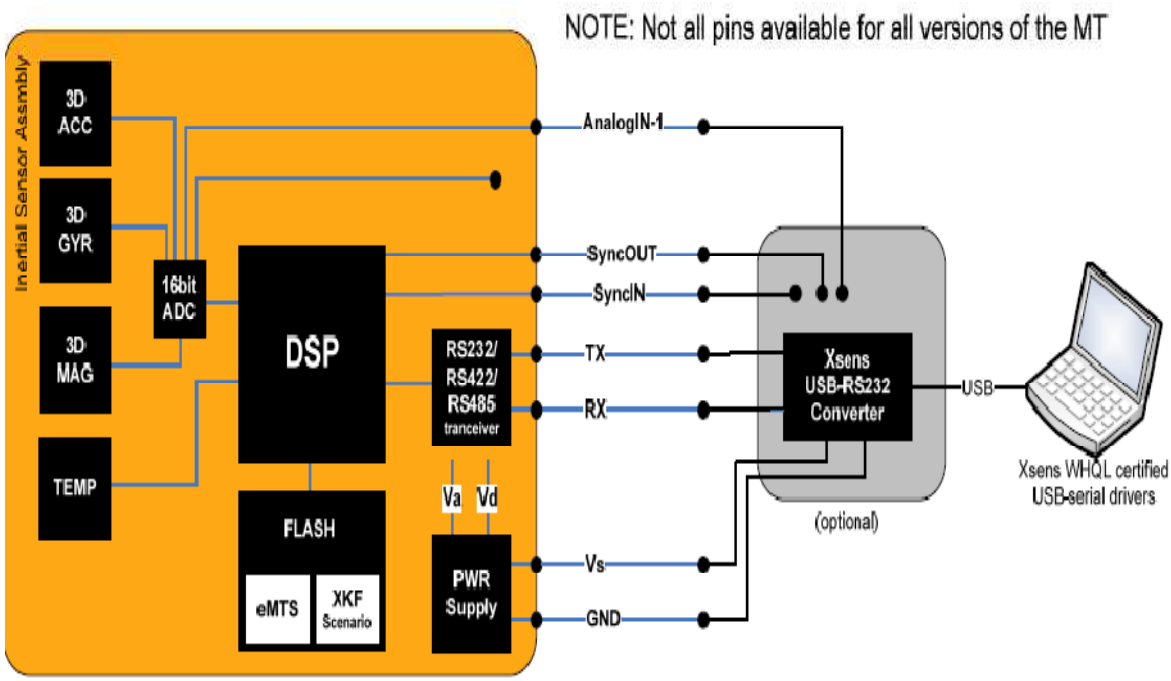

Figura 4: Conexionado del interior del sensor MTi

2.1.1.1 Xsens Kalman Filter del MTi

La orientación del MTi se calcula por el filtro de Kalman de 3 grados de libertad (XKF-3 por sus siglas en ingles). XKF-3 utiliza las señales de los giroscopios, acelerómetros y magnetómetros para calcular una estimación estadística óptima de la orientación en 3D en alta precisión sin la interferencia de los desvíos tanto estáticos como dinámicos.

El diseño del algoritmo XKF-3 puede ser explicado como un algoritmo de fusión sensorial donde la medición de la gravedad (por los acelerómetros 3D) y el norte magnético de la Tierra (por los magnetómetros 3D) se compensan mutuamente lentamente, pero sin límites, incrementando (por deriva) errores de la integración de datos de tasa de giro.

USO DE LA ACELERACIÓN DE LA GRAVEDAD PARA ESTABILIZAR LA MEDIDA DE LA INCLINACIÓN (roll / pitch)

XKF-3 estabiliza las medidas de inclinación (por ejemplo, combinando los ángulos roll y pitch conocemos la posición) utilizando las señales del acelerómetro. Un acelerómetro mide la aceleración de la gravedad, más la aceleración debida al movimiento del objeto con respecto a su entorno.

XKF-3 utiliza la hipótesis de que el promedio de la aceleración debida al movimiento es cero. Suponiendo esta hipótesis, la dirección de la gravedad puede ser observada y utilizada para estabilizar la medida de la posición. La clave aquí es la cantidad de tiempo durante la cual debe ser la aceleración promedio para que la hipótesis sea cierta. Durante este tiempo, los giroscopios deben ser capaces de seguir la orientación con un alto grado de precisión. En la práctica, esto limita la cantidad de tiempo durante el cual el supuesto es cierto. Para los giroscopios utilizados en el MTi, este período de tiempo es de aproximadamente 10-20 segundos como máximo.

USO DEL CAMPO MAGNETICO TERRESTRE PARA ESTABILIZAR EL HEADING (YAW)

Por defecto, el '*heading*' está estabilizado usando el campo magnético local**,** el campo magnético medido, se utiliza como una brújula. Si el campo magnético de la Tierra local se encuentra temporalmente perturbado, XKF-3 hará un seguimiento de esta alteración en vez de asumir incorrectamente que no hay perturbación. Sin embargo, en caso de perturbaciones estructurales magnéticas (> 10 a 20 segundos), el '*heading*' calculará lentamente una solución con el nuevo norte magnético local. Tenga en cuenta que el campo magnético no tiene efecto directo en la estimación de inclinación.

En el caso especial de que el MTi esté rígidamente atado a un objeto que contiene materiales ferromagnéticos, entonces las perturbaciones magnéticas estarán

presentes. El uso de un *'magnetic field mapping'* hace que las perturbaciones magnéticas sean eliminadas, permitiendo que el MTi pueda ser utilizado como si no estuviera unido a un objeto que contiene materiales ferromagnéticos.

INICIALIZACIÓN

El algoritmo XKF-3 no sólo calcula la orientación, sino que también realiza un seguimiento de variables tales como las alteraciones de los sensores o las propiedades del campo magnético. Por esta razón, la salida de orientación puede llevar cierto tiempo estabilizarse una vez que el MT se pone en modo de medición. El tiempo para obtener una salida estable óptima depende de una serie de factores. Un factor importante que determina el tiempo de estabilización es determinado por el tiempo para corregir pequeños errores en el sesgo de la tasa de giroscopios. Estos errores son debidos a cambio de temperatura o la exposición a los impactos. Para reducir el tiempo de estabilización, el último error calculado puede ser almacenado en la unidad del sensor de memoria no volátil. Si el MTi se usa después de un corto período de tiempo al apagar el giroscopio, por lo general no ha cambiado mucho y el tiempo de estabilización será típicamente menos de 10 segundos. Por otra parte, XKF-3 se activara más rápido y llegará más rápido a la zona óptima de funcionamiento si se inicia en un entorno sin perturbaciones magnéticas.

2.1.2 Comunicación Básica

Esta sección describe los conceptos básicos de cómo comunicarse con el MTi directamente, en bajo nivel, con comunicación RS-232 de serie, con el uso de un convertidor de Xsens USB-serie.

El protocolo de comunicación, que es el mensaje de base, permite al usuario cambiar la configuración del MTi y recuperar los datos desde el dispositivo. La configuración es totalmente configurable por el usuario, por ejemplo, frecuencia de muestreo, sincronización de la salida, velocidad de transmisión y modos de salida de datos, todo puede ser cambiado para adaptarse a las necesidades del usuario.

Todos los cambios de configuración deben realizarse mientras el dispositivo está en el llamado estado de configuración. En este estado el dispositivo acepta los mensajes que ajustan el modo de salida o cambia los otros parámetros. Siempre que la configuración esté completada, el usuario puede configurar el dispositivo para el estado de medida.

2.1.2.1 Estados

El MTi tiene dos estados que son:

- **Estado de configuración:**
  El estado de configuración se utiliza para obtener y/o configurar varios ajustes del sensor. La mayoría de los ajustes cambiarán la configuración que define la funcionalidad del dispositivo en el estado de la medición. Estos ajustes son, por ejemplo, la velocidad de transmisión de comunicación, período de muestreo, el modo de salida o las propiedades de sincronización.
  Antes de entrar en el estado de medición, el mensaje de configuración siempre se envía al controlador del sensor. Esta es la configuración que se lee desde la memoria interna no volátil y se utilizará en el estado de medición.

  Todos los ajustes se almacenan en un formato desarrollado por Xsens conocido como las siglas eMTS (‘*extended Motion Tracker Specification’*).

  Los ajustes modificados en el estado de configuración son inmediatamente almacenados en la memoria y se conserva su valor más reciente, incluso si se desconecta el dispositivo de su fuente de alimentación. Algunos mensajes tienen un parámetro adicional que requiere que el usuario especifique expresamente si los valores deben ser almacenados en una memoria no volátil. De cualquier manera, los cambios de configuración son inmediatos.

  Nota: Hay una excepción con la velocidad de transmisión. La nueva configuración no se usa inmediatamente, se usará en el próximo encendido o después de una señal de reset.

- **Estado de medición:**
  En este estado el dispositivo da salida al mensaje que contiene los datos de la configuración actual. Un solo mensaje, ‘*MTData*’, se utiliza para todos los modos de salida diferentes. Por ello es importante que el controlador del sensor sepa cómo está configurado el dispositivo. La configuración actual determinará cómo los datos del mensaje deben de ser interpretados. Un mensaje especial, ‘*Configuration*’, contiene la información pertinente con los datos recibidos para que en el estado de medición el controlador pueda interpretarlos de forma inequívoca. En el registro del mensaje ‘*MTData*’ es recomendable incluir el mensaje de configuración en la cabecera de datos para futuros análisis o post-procesamiento.

La configuración que tiene por defecto el MTi se muestra en la Tabla 2.

Tabla 2: Configuración por defecto del MTi

Propiedad	Valor
Modo de salida	Salida de la orientación
Opciones de salida	Orientación en el modo cuaternios Contador de muestreo habilitado
Frecuencia de muestreo	100 Hz
Baudios	115200 bps
Salida del skip factor	0

Hay dos maneras de entrar en el estado de configuración o en el estado de medición. En el arranque o después de una señal de reset, el dispositivo empieza con el proceso '*WakeUp*' (activación), si el controlador no responde al mensaje, entrará en el estado de medición con la última configuración almacenada. Pero si se envía el mensaje '*WakeUpAck*' antes de 500ms desde la recepción del '*WakeUp*' entrará en el estado de configuración.

La otra forma de entrar en el estado de configuración o medición es usando los mensajes '*GoToConfig*' o *'GoToMesasurement'* mientras que esté el otro estado activo.

En la Figura 5 se muestra un esquemático de los estados del sensor y el modo en el que se puede conmutar entre ellos.

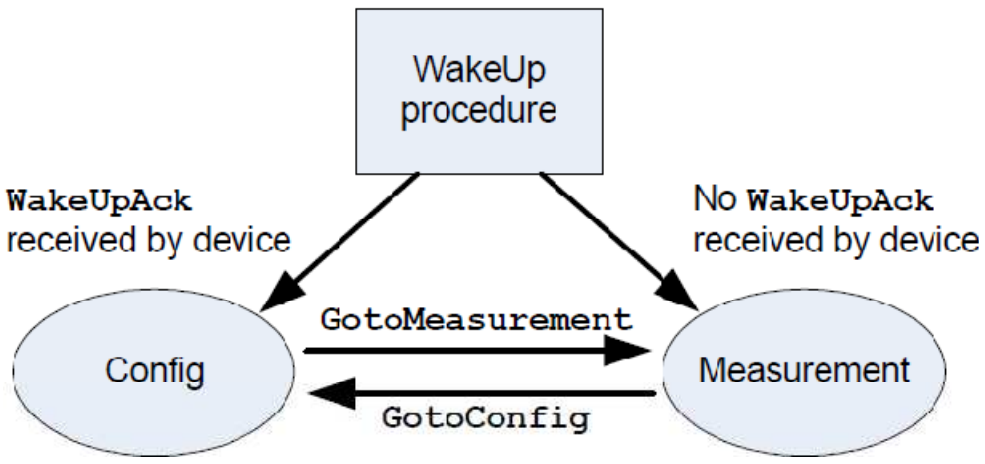

Figura 5: Estados del sensor

2.1.2.2 Mensaje

La comunicación con el MTi se hace por medio de mensajes, que se construyen de acuerdo a una estructura estándar. El mensaje MTi estándar puede contener desde 0 a 254 bytes de datos.

Un mensaje MTData contiene los seis tipos de campos, que se muestran en la Figura 6, los cuales se detallarán en la Tabla 3 que se muestra a continuación.

Figura 6: Campos del mensaje MTData

Tabla 3: Especificación del los campos del mensaje

Campo	Valor del campo	Descripción
PRE	1 byte	Cabecera, indicador de inicio de paquete → 250 (0xFA)
BID	1 byte	Identificador del bus / dirección → 255 (0xFF)
MID	1 byte	Identificador del mensaje
LEN	1 byte	Valor del número de bytes en el campo DATA. Valor máximo es 255 (0xFE). El valor 255 (0xFF) está reservado.
DATA	0 – 254 bytes	Bytes de datos (opcional)
CS	1 byte	Mensaje de comprobación

- **Cabecera (PRE)**

Cada mensaje comienza con la cabecera que siempre contiene el valor 250 (=0xFA **).**

- **Identificador de bus (BID) o la dirección**

El campo BID está incluido en el formato de los mensajes para que sean compatibles con la XbusMaster que se conecta a múltiples MT. En este caso utilizaremos el valor de dirección 255 (0xFF) que indica un "dispositivo maestro".

- **Identificador de mensaje (MID)**

Este campo identifica el tipo de mensaje. Más adelante se detallará una lista de los tipos existentes.

- **Longitud (LEN)**

Especifica el número de bytes de datos en el campo DATA. El valor 255 (= 0xFF) está reservado. Esto significa que un mensaje tiene una carga útil máxima de 254 bytes. Si la longitud es cero no existe campo de datos.

- **Datos (DATA)**

Este campo contiene los bytes de datos y tiene una longitud variable que se especifica en el campo de longitud. La interpretación de los bytes de datos del mensaje depende del modo de salida.

- **Suma de comprobación (CS)**

Este campo se utiliza para la detección de errores. Si se suman todos los bytes del mensaje excepto la cabecera y el valor del byte inferior del resultado es igual a cero, el mensaje es válido y se puede procesar. El valor de suma de comprobación del mensaje debe ser incluido en la suma.

Forma de hacer la comprobación:

1. Pasamos a decimales los datos
2. Los sumamos
3. Calculamos el resto a 256
4. Restamos el resto a 256

2.1.2.3 Tiempo de comunicación

Para muchas aplicaciones pueden ser cruciales saber exactamente las diversas demoras en el sistema. En esta sección se describe cómo se relaciona el tiempo entre eventos físicos y el dispositivo de salida en los modos de uso básico del MTi. En la Figura 7 se muestran los módulos del sensor MTi.

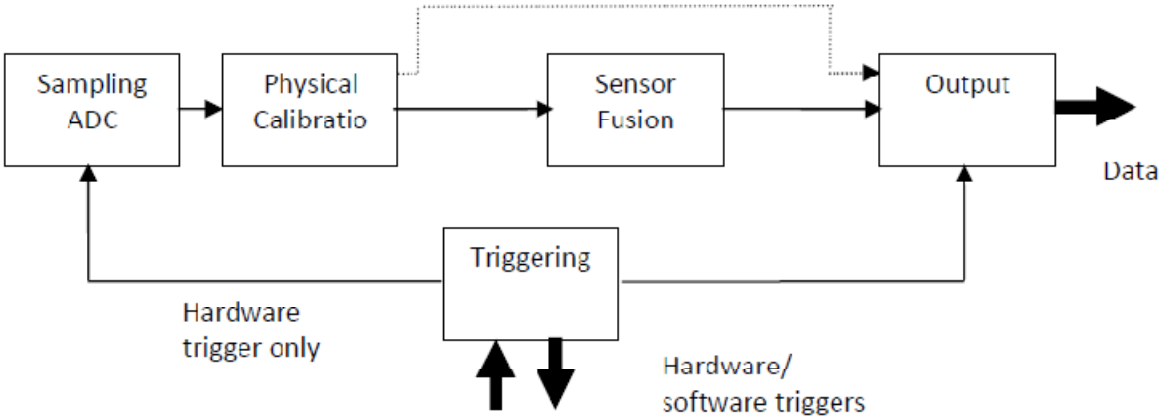

Figura 7: Bloques del sensor

Cuando el MTi está en el estado de medición, el DSP (procesador digital de señales) interno está continuamente ejecutando un bucle como el diagrama anterior. El disparo puede ser generado por muestreo del dispositivo interno, de software externo, o incluso de hardware (por lo general no se recomienda).

El retraso de tiempo entre un evento físico (por ejemplo, un cambio de orientación o de aceleración) es dictado por dos factores:

1. Adquisición interna y tiempo de cálculo.
2. Tiempo de la transmisión serie.

La adquisición interna y el tiempo de cálculo depende de la situación y el modo de salida. En la Tabla 4 se muestra la adquisición interna y los tiempos de cálculo de todos los escenarios y los diferentes modos de salida. Dado que el '*Xsens Kalman Filter*' se aplica en función de los diferentes cálculos de los datos disponibles, el tiempo de cómputo no es constante. El tiempo de adquisición interno de los datos en bruto es 0.19 ms.

Tabla 4: Tiempos de calibración

Escenario + modo de salida	Adquisición de datos calibrados y cálculo de tiempo	Peor caso de adquisición de la orientación y cálculo de tiempo
XKF-3 Escenario - Humano	0.31 ms	2.55 ms
XKF-3 Escenario – Humano aceleraciones grandes	0.31 ms	2.55 ms
XKF-3 Escenario – Máquina	0.31 ms	2.55 ms
XKF-3 Escenario – Máquina '*nomagfield*'	0.31 ms	2.01 ms
XKF-3 Escenario - Marina	0.31 ms	2.55 ms

No hay diferencia entre la adquisición interna y el tiempo de cálculo de los datos de la orientación y los datos de calibrado/orientación juntos.

El tiempo de la transmisión serie se puede calcular fácilmente con la formula 2.1

$$\frac{bytes\ totales\ del\ mensaje * 10\ {}^{bits}/_{byte}}{baudios\ en\ la\ comunicación\ {}^{bits}/_{s}} = Tiempo\ de\ transmisión \quad (\mathbf{2.1})$$

2.1.2.4 Disparo y sincronización

En caso de utilizar múltiples sistemas durante una medición, es importante contar con los datos de medición sincronizada entre los sistemas. El procesamiento de datos sincronizados es mucho más fácil porque no hay necesidad de volver a muestrear los datos para compensar las imprecisiones del reloj. La sincronización con múltiples

sistemas implica dos cuestiones importantes: el comienzo de la medición al mismo tiempo y la relación de tiempo fijo de las instancias de toma de muestras.

El MTi tiene la capacidad de ser activado por dispositivos externos o accionar dispositivos externos para sincronizarse.

2.1.3 Especificación de salida de datos

Hay varios modos de salidas diferentes del MTi. Los dos modos principales, orientación de la salida y calibración de la salida de datos, se analizan por separado. Sin embargo, tenga en cuenta que los dos modos de salida se pueden combinar fácilmente, de modo que se consigue un paquete de datos, que contiene datos de orientación y datos inerciales calibrados, al mismo tiempo.

2.1.3.1 Sistema de coordenadas

Para la captación de los datos del sensor se tiene que tener claro los sistemas de coordenadas, tanto del sensor, como el de referencia, por eso, se explican a continuación.

CALIBRADO DEL SENSOR

Todas las lecturas del sensor calibrado (aceleraciones, velocidad de giro, campo magnético) están en el sistema cartesiano de coordenadas, tal como se define en la Figura 8. Este sistema de coordenadas es fijo al cuerpo del aparato, y se define como el sistema de coordenadas del sensor (S). La salida de la orientación 3D se discute más adelante.

.

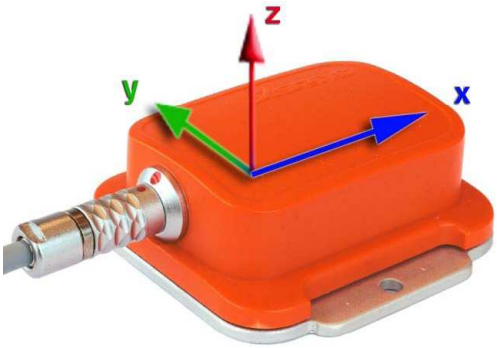

Figura 8: Sensor MTi con su sistema de coordenadas fijo (S)

La carcasa externa y la placa base de aluminio del MTi está cuidadosamente alineada con la salida del sistema de coordenadas. La alineación del plano del fondo y los lados de la plataforma de la base de aluminio con respecto a la salida del sensor, es de 0,1° del sistema de coordenadas fijo (**S**).

La alineación entre la carcasa (de plástico) y el sistema de coordenadas fijo (**S**) de la salida del sensor, no es de alta precisión. La alineación real entre el sistema de coordenadas (**S**) y la parte inferior de la carcasa de plástico está garantizada que es menos de 3°.

La no ortogonalidad entre los ejes del sistema de coordenadas fijo (**S**) y la carcasa es menor de 0,1°. Esto también significa que la salida de la aceleración lineal en 3D, el giro en 3D y los datos del campo magnético en 3D tendrán lecturas ortogonales XYZ con un error menor de 0,1°.

ORIENTACION DEL SISTEMA DE COORDENADAS

El MTi calcula la orientación entre el sistema de coordenadas fijo del sensor (**S**) y un sistema de coordenadas fijo de referencia a la Tierra (**G**).

Por defecto, el sistema de coordenadas fijo de referencia a la Tierra utilizado se define como un sistema de coordenadas cartesianas con:

- X positivo cuando apunta al norte magnético local.
- Y positivo cuando apunta al Oeste.
- Z positivo cuando apunta hacia arriba.

La salida de la orientación 3D (independiente del modo de salida) se define como la orientación entre el sistema de coordenadas fijo al sensor (**S**) y el sistema de coordenadas fijo a la Tierra (**G**) (mostrado en la Figura 9), utilizando el sistema de coordenadas fijo a la Tierra (**G**) como referencia del sistema de coordenadas.

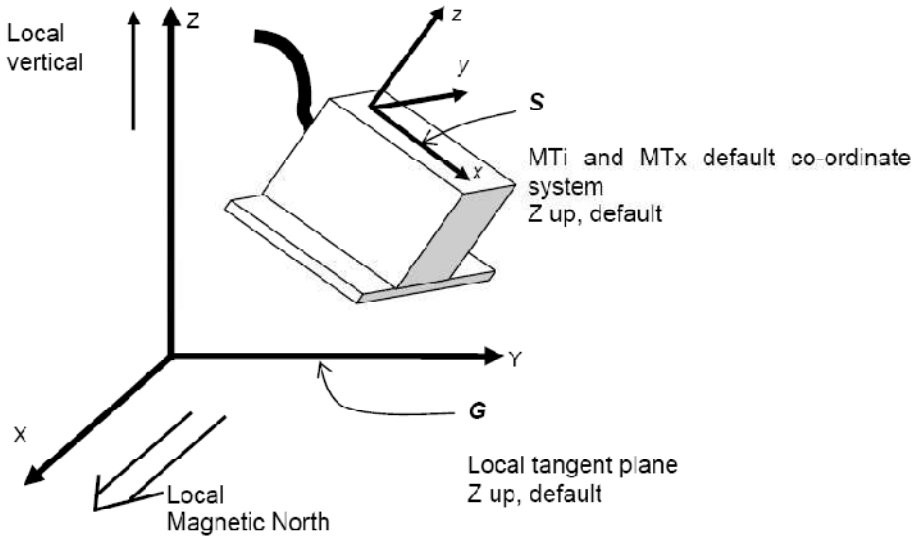

Figura 9: MTi en el sistema de coordenadas de la Tierra

La salida de sistema de coordenadas del MTi está referenciado respecto al norte magnético local. La desviación entre el norte magnético y el norte verdadero (conocida como la declinación magnética) varía dependiendo de su ubicación en la tierra y pueden ser obtenidos a partir de unos modelos diferentes del campo magnético de la Tierra como una función de la latitud y longitud. El MTi puede aceptar un ajuste del valor de la declinación. Esto se hace mediante el establecimiento de la declinación en el Administrador de MT, por el kit de desarrollo software (SDK, por sus siglas en inglés) o por comunicación directa con el sensor. Entonces la salida se verá compensada por la declinación y por lo tanto hará referencia al norte verdadero "local".

ESPECIFICACION DEL RENDIMIENTO DE LA ORIENTACION

Las características típicas del rendimiento del MTi de la salida de orientación son:

- Rango dinámico: todos los ángulos en 3D
- Resolución angular: 0.05°
- Repetitividad: 0.2°
- Precisión estática (‘*roll*’ / ‘*pitch*’): 0.5°
- Precisión estática (‘*yaw*’): 1.0°
- Precisión dinámica: 2° RMS
- Frecuencia: configurable por el usuario, máximo de 120 Hz

2.1.3.2 Modos de salida de orientación

La orientación calculada por el MTi es la orientación del sistema de sensores fijos de coordenadas (**S**) con respecto a un sistema de coordenadas cartesianas fijo a la Tierra (**G**). La orientación de salida se puede presentar en diferentes parametrizaciones:

Cuaternios Unidad (también conocidos como parámetros de Euler)

Un vector cuaternio unitario puede ser interpretado por la representación de la rotación alrededor de un vector unitario (***n)*** en un ángulo (***α***). Ver ecuación 2.2.

$$q_{GS} = \left(\cos\left(\frac{\alpha}{2}\right), \boldsymbol{n} sin\left(\frac{\alpha}{2}\right)\right) \quad \textbf{(2.2)}$$

Y se puede representar como en la ecuación 2.3:

$$q_{GS} = (q_0, q_1, q_2, q_3) \qquad \textbf{(2.3)}$$

La definición de salida en el modo cuaternio es la representada en la Figura 10.

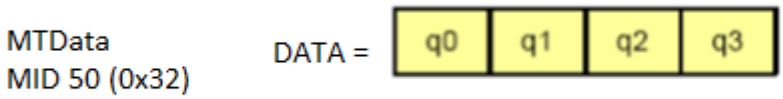

Figura 10: Composición del mensaje MTData con cuaternios

Todos los datos del elemento DATA son de tipo FLOATS (4 bytes), salvo que se especifique otro tipo de formato mediante la modificación de la función "OutputSetting Data Format"

Ángulos de Euler

Los ángulos de Euler constituyen un conjunto de tres coordenadas angulares, que sirven para especificar la orientación de un sistema de referencia de ejes ortogonales, normalmente móvil, respecto a otro sistema de referencia de ejes ortogonales normalmente fijos.

Tres ángulos, que mediante una sucesión ordenada de giros, definen el cambio de un sistema de coordenadas a otro. Los ángulos de Euler (phi, theta, psi) corresponden con los ángulos convencionales de '*roll*' (**Φ**), '*pitch*' (**θ**), '*yaw*' (**ψ**) que se utilizan en navegación para especificar la actitud de un móvil. Poniendo el ejemplo de un avión, como se puede observar en la Figura 11, el ángulo de '*roll*' (**Φ**) es el giro de las alas del avión, el ángulo de '*pitch*' (**θ**) la inclinación del morro y el ángulo de '*yaw*' (**ψ**) es el giro del morro del avión respecto al norte [12].

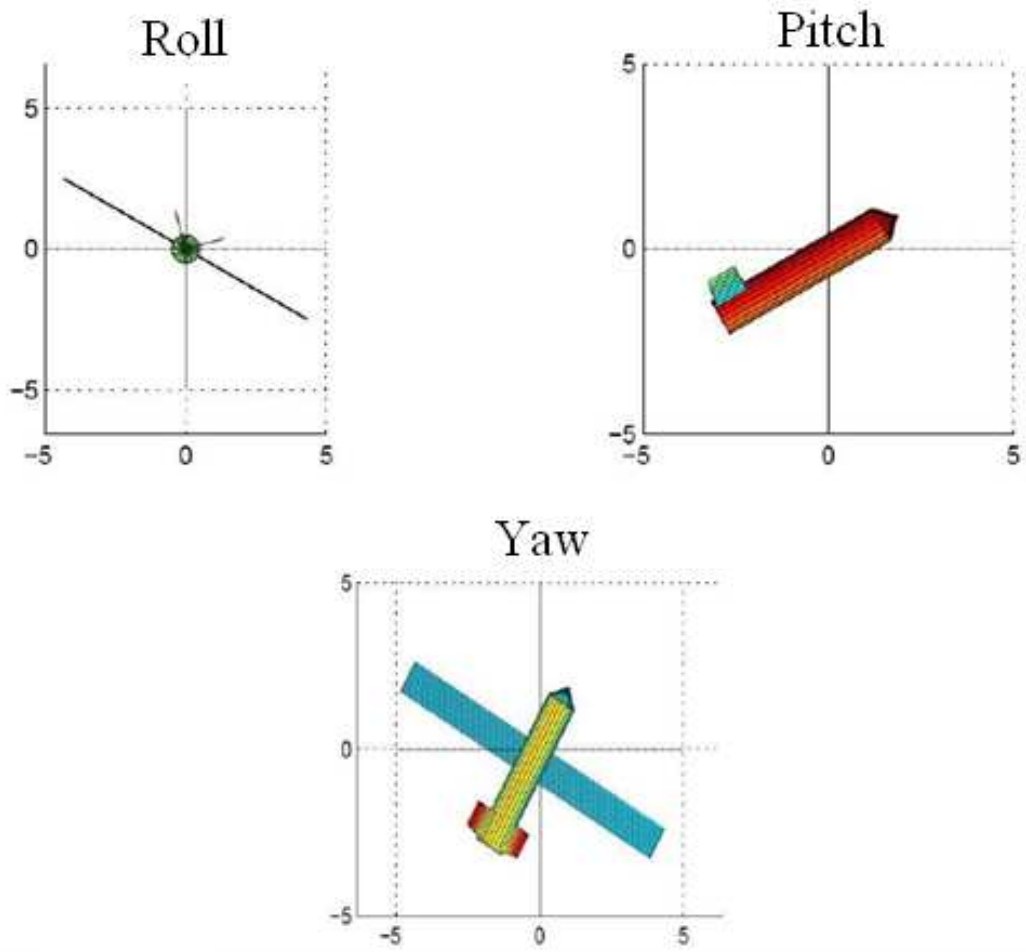

Figura 11: Ángulos de Euler

Es importante señalar que los ángulos de Euler no están únicamente definidos y existen ambigüedades. Por eso se deben hacer las rotaciones siempre en el mismo orden ya que los mismos ángulos, si son aplicados con órdenes diferentes, dan lugar a transformaciones diferentes.

- Φ = '*roll*' = rotación alrededor del eje X_G, definido por [-180º a 180º].
- θ = '*pitch*' = rotación alrededor del eje Y_G, definido por [-90º a 90º].
- ψ = '*yaw*' = rotación alrededor del eje Z_G, definido por [-180º a 180º].

La salida de los datos está en grados y no en radianes.

La definición de salida en el modo de orientación para los ángulos de Euler se representa en la Figura 12.

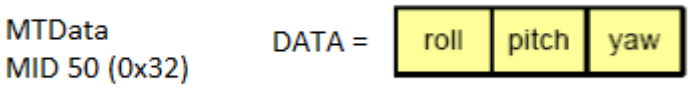

Figura 12: Composición del mensaje MTData con ángulos de Euler

Todos los datos del elemento DATA son de tipo FLOATS (4 bytes), salvo que se especifique otro tipo de formato mediante la modificación de la función "OutputSetting Data Format".

Matriz de rotación

La matriz de rotación (también conocida como '*Direction Cosine Matrix*', DCM) es una representación muy conocida, redundante y completa de la orientación. La matriz de rotación se puede interpretar como los componentes del vector unitario del sistema de coordenadas del sensor (***S***) expresado en ***G***. Para el vector unitario R_{GS} de ***S*** se encuentra en las columnas de la matriz, por lo que la col 1 es X_S expresado en ***G***, etc. La normal de la matriz de rotación es siempre igual a uno (1) y una rotación de R_{GS} seguido por el inverso de rotación R_{SG}, se obtiene la matriz identidad I^3. Se puede observar en la ecuación 2.4.

$$||\ R\ || = 1 \qquad R_{GS}\, R_{SG} = I^3 \qquad (\mathbf{2.4})$$

La matriz de rotación, $R_{GS,}$ puede ser interpretada por los términos de los cuaternios como se representa en la ecuación 2.5.

$$R_{GS} = \begin{bmatrix} 2q_0^2 + 2q_1^2 - 1 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 2q_0^2 + 2q_2^2 - 1 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 2q_0^2 + 2q_3^2 - 1 \end{bmatrix} \quad (\mathbf{2.5})$$

También, la matriz de rotación se puede expresar por los ángulos de Euler como se muestra en la ecuación 2.6.

$$R_{GS} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \quad (\mathbf{2.6})$$

Para una mayor comodidad de notación, la matriz, $R_{GS,}$ se puede representar como se muestra en la ecuación 2.7.

$$R_{GS} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \quad (\mathbf{2.7})$$

La composición del mensaje de salida, MTData, en este modo se muestra en la Figura 13 cuyos datos son los componentes de la matriz descrita anteriormente en 2.7.

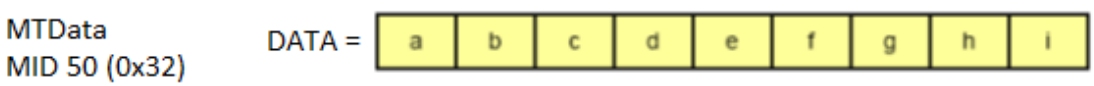

Figura 13: Composición del mensaje MTData para el modo rotación

Todos los datos del elemento DATA son de tipo FLOATS (4 bytes), salvo que se especifique otro tipo de formato mediante la modificación de la función "OutputSetting Data Format".

2.1.3.3 Modos de salida de los datos calibrados

Esta sección está dedicada a dar información detallada sobre la definición de los modos de salida de datos inerciales calibrados del MTi

FISICA DEL SENSOR

Los sensores físicos dentro del MTi (acelerómetros, giroscopios y magnetómetros) están calibrados de acuerdo a un modelo físico de la respuesta de los sensores a las diferentes magnitudes físicas, por ejemplo, la temperatura. El modelo básico es lineal y de acuerdo con la ecuación 2.8.

$$s = K_T^{-1}(\boldsymbol{u} - \boldsymbol{b}_T) \qquad (2.8)$$

El modelo utilizado realmente es más complicado y se están desarrollando constantemente nuevos métodos. A la calibración de cada MTi fabricado se le ha asignado una matriz de ganancia única ($\mathbf{K_T}$), y el vector de polarización ($\mathbf{b_T}$). Estos datos de calibración se utilizan para relacionar las tensiones de la muestra digital (**u**), (enteros sin signo de 16 bit, salida de un conversor analógico-digital) de los sensores a las respectivas cantidades físicas (**s**).

En la ecuación 2.9 se representa la matriz de ganancia, que se divide en una matriz de desalineación (**A**) y en una matriz de ganancia (**G**). La desalineación determina la dirección de los ejes con respecto a los ejes del sistema de coordenadas de la carcasa del sensor (**S**). Por ejemplo, el primer elemento de la matriz de desalineación es el acelerómetro ($\mathbf{a_{1,x}}$) que describe la sensibilidad a la dirección del acelerómetro en el canal uno. La '**O**' representa los modelos de mayor orden y modelos de temperatura, correcciones de sensibilidad gravedad, etc.

$$A = \begin{bmatrix} a_{1,x} & a_{1,y} & a_{1,z} \\ a_{2,x} & a_{2,y} & a_{2,z} \\ a_{3,x} & a_{3,y} & a_{3,z} \end{bmatrix} \qquad G = \begin{bmatrix} G_1 & 0 & 0 \\ 0 & G_2 & 0 \\ 0 & 0 & G_3 \end{bmatrix} \tag{2.9}$$

$$K_T = \begin{bmatrix} G_1 & 0 & 0 \\ 0 & G_2 & 0 \\ 0 & 0 & G_3 \end{bmatrix} \begin{bmatrix} a_{1,x} & a_{1,y} & a_{1,z} \\ a_{2,x} & a_{2,y} & a_{2,z} \\ a_{3,x} & a_{3,y} & a_{3,z} \end{bmatrix} + \mathbf{O}$$

La salida calibrada de la aceleración lineal en 3D, el índice de vuelta en 3D (giro) y los datos del campo magnético en 3D están en el sistema de coordenadas fijo del sensor (**S**).

Las unidades de la salida de los datos calibrados por los sensores son las mostradas en la Tabla 5.

Tabla 5: Unidades de los datos calibrados

Vector	Unidad
Aceleración	m/s^2
Velocidad angular (velocidad de giro)	rad/s
Campo magnético	a. u. (unidades arbitrarias) normalizado a la intensidad del campo de la Tierra

Los datos calibrados están "sin procesar", es decir, sólo el modelo de calibración física se aplica a los valores de 16-bit encontrado en los convertidores analógico-digital. No hay un filtrado adicional u otro proceso temporal aplicado al dato.

La composición de mensaje MTData en el modo de salida de datos calibrados es el mostrado en la Figura 14. Los tres primeros datos (accX, accY y accZ) corresponden a las aceleraciones en cada eje, los siguientes (gyrX, gyrY y gyrZ) a la velocidad angular y los tres últimos (magX, magY y magZ) al campo magnético en cada uno de los ejes correspondientes.

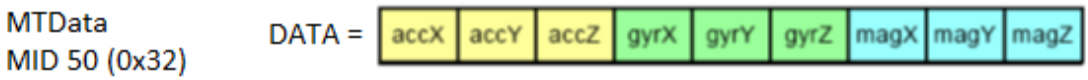

Figura 14: Composición del mensaje MTData para modo de salida calibrado

Todos los datos del elemento DATA son de tipo FLOATS (4 bytes), salvo que se especifique otro tipo de formato mediante la modificación de la función "OutputSetting Data Format"

2.1.3.4 Modo de salida sin calibrar

El formato de salida no calibrada pura, significa que el modelo de calibración física descrito en el apartado anterior no se aplica. Esto le da libre acceso al nivel básico de la unidad del sensor, pero en la mayoría de los casos este nivel de uso no se recomienda. Sin embargo, si su objetivo principal es el registro y el procesamiento posterior, puede ser ventajoso ya que siempre es posible volver a la "fuente" de la señal. En este modo la temperatura del dispositivo también se envía como se muestra en la Figura 15 cuya estructura es idéntica a la Figura 14, solo que se añade el campo "temp" donde se almacena la temperatura del dispositivo.

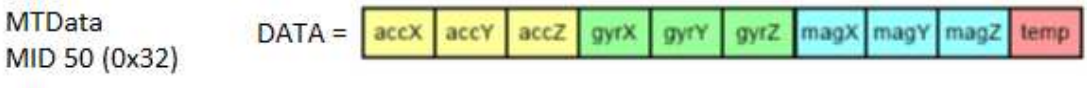

Figura 15: Composición del mensaje MTData para modo de salida sin calibrar

Cada elemento de datos en el campo DATA es de 2 bytes (16 bits) enteros sin signo.

Los 2 bytes de datos de la temperatura en el modo de salida no calibradas del MTi se puede interpretar como 16 bits con complemento a 2.

2.1.4 Estructura del Software

2.1.4.1 Niveles CMT

En esta sección se describe el concepto de las interfaces CMT en cada nivel, en la Figura 16 se muestra un esquema de los niveles para una mejor visualización.

CMT Nivel 1 - Interfaz hardware

Las funciones de este nivel dan acceso a las comunicaciones de entrada/salida de bajo nivel. Estas funciones son dependientes de la plataforma (sistema operativo y procesador) por lo que pueden tener que ser modificadas para aplicaciones integradas. El propósito de separar a este nivel de los otros niveles CMT es para que los programadores de dispositivos integrados puedan facilitar las modificaciones de código y recompilar la biblioteca hasta CMT Nivel 3 y que puedan utilizar la funcionalidad en alto nivel inmediatamente.

CMT Nivel 2 – Interfaz de mensaje

La interfaz mensaje reúne el mensaje de protocolo comunicación MT usando las funciones del nivel 1, que retorna datos de mensaje en bruto. Si la suma de comprobación es correcta, el mensaje completo se transmite al siguiente nivel CMT. Este nivel tiene su propio tiempo de espera (s) que es independiente del nivel 1, salvo que se hayan adoptado disposiciones que entonces si serán más largos que el nivel 1.

La CMT Nivel 2 interfaz no utiliza ningún otro método específico de direccionamiento más que el que introduce y utiliza el protocolo de comunicación MT.

Normalmente, no hay necesidad de utilizar directamente la interfaz de la CMT o modificar el nivel 2.

CMT Nivel 3 – Interfaz de dispositivo

La CMT Nivel 3 implementa las siguientes capacidades:

- Mantiene el registro de las configuraciones de los dispositivos MT para un solo dispositivo Máster MT (puerto de comunicaciones o archivo)
- Este nivel implementa todos los mensajes MT. Por ejemplo gotoConfig, setDeviceMode, getProductCode, etc.
- MT DATA se recupera en una estructura de paquetes especial que interpreta y traduce el contenido de los datos para acceso de datos fácil.

- Este nivel también tiene función de registro, que le permite registrar todos los mensajes recibidos de forma automática y luego leer el archivo.
- Los dispositivos son tratados por DeviceID y no por el número de puerto serie.

CMT Nivel 3 está disponible para los desarrolladores como el componente de más alto nivel de la biblioteca estática y como código fuente.

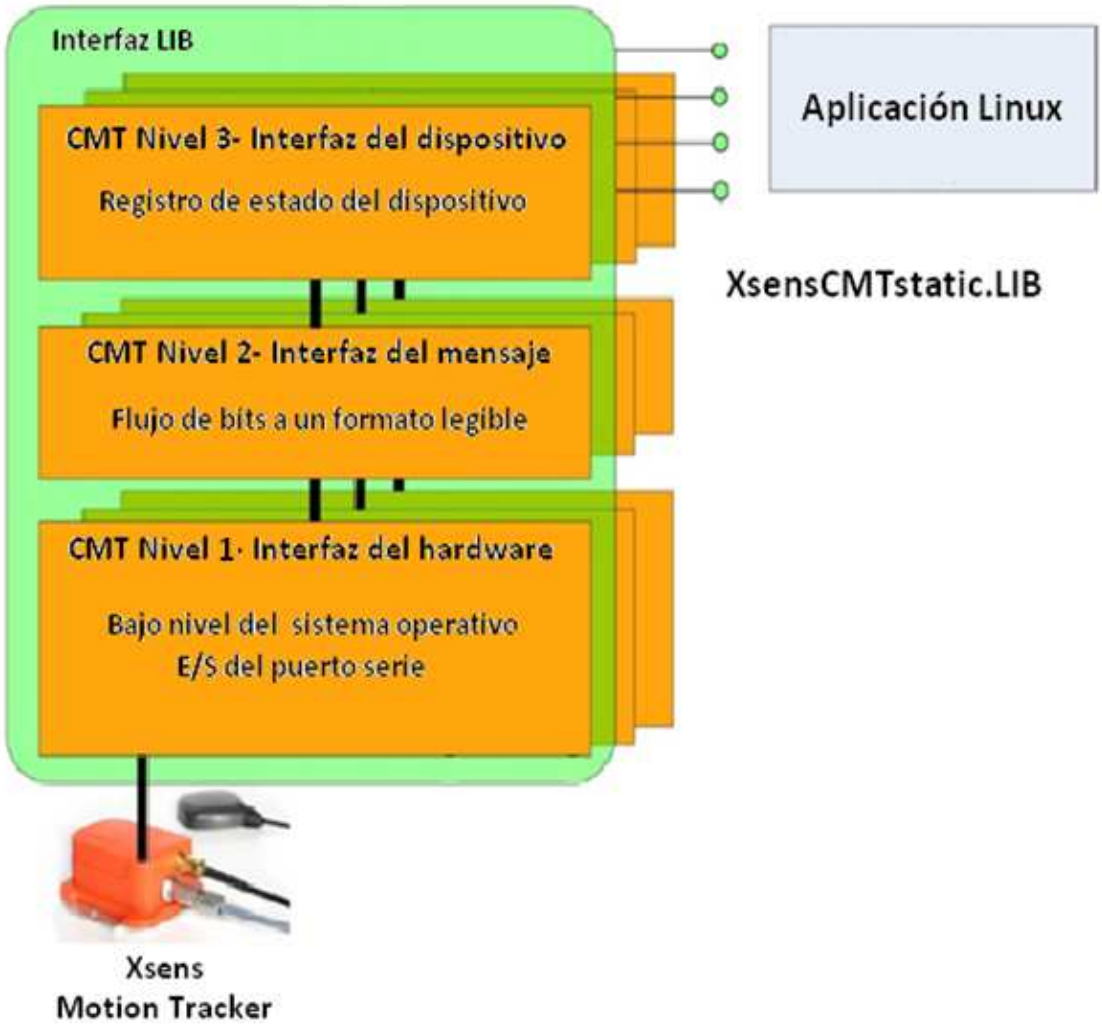

Figura 16: Niveles CMT

2.1.4.2 Descripción de las funciones CMT

CMT contiene grupos de funciones que pueden llevarse a cabo bajo el modo de configuración (cmtGotoConfig), el modo de medición (cmtGotoMeasurement) o en cualquier modo. Muchas de estas funciones llevan el número de petición y DeviceID como parámetros comunes. Esto informa al CMT de que objeto CMT usar y por supuesto, el DeviceID está unido a un dispositivo Xsens específico.

- **Inicializar y desactivar:** Utilizar estas funciones para inicializar el CMT para el manejo de dispositivos o archivos.
  - **cmtCreateInstance:** Crea una petición de un objeto CMT. Cada instancia puede manejar uno o más archivos o de lo contrario uno o más puertos COM. El parámetro "serialNumber" debe contener un número de serie válido que se suministra con la compra del dispositivo.
  - **cmtDestroyInstance:** Destruye el objeto CMT dado, la liberación de su número de petición para su uso por otro proceso. Se va a cerrar todos los archivos y los puertos asociados a ella.
  - **cmtClose:** Cerrar todos los dispositivos conectados a la petición determinada.
  - **cmtOpenPort**: Abre el puerto COM especificado.
  - **cmtClosePort**: Cierra el puerto COM especificado.
  - **cmtOpenLogFile:** Abre un archivo de registro.
  - **cmtScanPorts:** Escanea los puertos conectados al dispositivo Xsens.

- **Configuración de funciones Get y Set:** Recupera y establece los parámetros de configuración de un dispositivo de Xsens. Estas funciones se refieren específicamente a un dispositivo Xsens por ID:

  GET()

  - **cmtGetBaudrate:** Obtener la velocidad de transmisión que se utilizan actualmente.
  - **cmtGetConfiguration:** Obtener la configuración completa del dispositivo.
  - **cmtGetDeviceId:** Obtiene la ID (identificación) del dispositivo.
  - **cmtGetDroppedPacketCount:** Obtener el número de paquetes perdidos durante la medición.
  - **cmtGetMainDeviceCount:** Obtener el número de dispositivos maestros conectados o **0** si no está conectado.
  - **cmtGetMtCount:** Obtener el número de MT conectados.
  - **cmtGetMainDeviceId:** Obtener la ID de la electrónica principal asociado con el objeto CMT.
  - **cmtGetDeviceMode**: Obtener el modo completo de salida del dispositivo.
  - **cmtGetErrorMode:** Obtener el modo de error.
  - **cmtGetHeading:** Obtener el offset de un dispositivo. El rango es-pi a +pi.

- **cmtGetLocationId:** Obtener la ubicación del ID de un dispositivo.
- **cmtGetMagneticDeclination:** Obtener la declinación magnética almacenada. El rango es -pi a +pi.
- **cmtGetPortNr:** Obtener el puerto en el que un dispositivo específico está conectado.
- **cmtGetSampleFrequency:** Obtiene la frecuencia a la que está conectado el dispositivo.
- **cmtGetSerialBaudrate:** Obtenerla velocidad de transmisión que se reporta para la comunicación serie de un puerto.
- **cmtGetAvailableScenarios:** Obtener una lista de los escenarios disponibles. Esta función puede devolver más escenarios que están presentes en el sensor específico, ya que el software CMT puede definir escenarios adicionales
- **cmtGetScenario:** Obtener el escenario que se está usando actualmente por el dispositivo indicado.
- **cmtGetGravityMagnitude:** Obtener la magnitud que actualmente utiliza el vector de la gravedad.
- **cmtGetLatLonAlt:** Obtener la actual latitud/ longitud/ altitud.
- **cmtGetObjectAlignmentMatrix:** Obtener la matriz objeto de alineación que se utiliza actualmente.
- **cmtGetTransmissionDelay:** Obtener el retardo de la transmisión que se utiliza actualmente.

SET()

- **cmtSetBaudrate:** Ajusta la velocidad de transmisión y vuelve a conectarse a la nueva velocidad.
- **cmtSetDeviceMode:** Establece el modo de salida de un dispositivo.
- **cmtForceSetDeviceMode:** Establece el modo de salida completa de un dispositivo. Se escribe siempre el modo, la configuración y la frecuencia de muestreo.
- **cmtSetErrorMode:** Establece el modo de error. El modo de error define la forma en que el dispositivo debe tratar la falta de mensajes de errores. Por defecto, si una muestra se pasa por alto, el contador de la muestra se incrementa y si no se realiza ninguna acción ERRORMODE = 1.
- **cmtSetQueueMode:** Establece el modo de cola (FIFO o LAST).
- **cmtSetHeading:** Establecer el '*heading*' del dispositivo dado. El rango válido es -pi api.
- **cmtSetMagneticDeclination:** Ajusta la declinación magnética almacenada (radianes entre-pi y pi).
- **cmtSetScenario:** Especifica el escenario que se utilizará para el procesamiento de datos.
- **cmtSetGravityMagnitude:** Define la magnitud del vector de la gravedad. Por defecto es 9,81, pero puede ser alterada para fines especiales, tales como las aplicaciones espaciales o para experimentos de casi caída libre.

- **cmtSetLatLonAlt:** Especifica la actual latitud/ longitud/ altitud.
- **cmtSetObjectAlignmentMatrix:** Especifica la matriz objeto de alineación que se utiliza actualmente.
- **cmtSetTransmissionDelay:** Establece el retardo de la transmisión que se utiliza actualmente.
- **cmtSetNoRotation:** Especifica la duración de "no hay rotación".

- **Funciones de manejo de datos:** Estas funciones son válidas en el modo de medición.
  - **cmtGetNextDataBundle:** Recupera un mensaje de datos. Especifique si desea obtener los datos en una FIFO o en la última forma de datos.
  - **cmtDataGetRawData:** Obtiene los componentes de datos en bruto, de un elemento de datos.
  - **cmtDataContainsRawData:** Comprueba si el elemento de datos contiene datos sin procesar.
  - **cmtDataGetCalData:** Obtiene el componente de datos de calibración de un elemento de datos.
  - **cmtDataContainsCalData:** Comprueba si el elemento de datos contiene los datos de calibración.
  - **cmtDataGetOriEuler:** Obtiene los componentes de la orientación en ángulos de Euler.
  - **cmtDataContainsOriEuler:** Comprueba si el elemento de datos contiene los datos de orientación de Euler.
  - **cmtDataGetOriMatrix:** Obtiene los componentes de la orientación en una matriz de orientación.
  - **cmtDataContainsOriMatrix:** Comprueba si el elemento de datos contiene los datos de la matriz de orientación.
  - **cmtDataGetSampleCounter:** Obtiene el contador de muestras de los componentes del paquete.
  - **cmtDataGetAccG:** Obtiene el componente XKFAcc-3-G del paquete.

- **Funciones de Control de archivos:** Estas funciones son válidas en el modo de medición.
  - **cmtOpenLogFile:** Abre un archivo de registro de entrada.
  - **cmtCloseLogFile:** Cierra un archivo de registro abierto.
  - **cmtCreateLogFile:** Crear un archivo de registro para un puerto determinado.
  - **cmtCloseLogFileForPort:** Cierra un archivo de registro abierto que se asocia a un puerto determinado.
  - **cmtGetLogFileReadPos:** Recupera el archivo de registro actual y la posición de lectura del archivo de registro.

- **cmtGetLogFileSize:** Obtiene el tamaño del archivo de registro.
- **cmtResetLogFileReadPos:** Restablece la posición de lectura del archivo de registro.
- **cmtSetLogMode:** Activa o desactiva el registro de datos en un archivo.

- **Funciones Genéricas**
  - **cmtGetFirmwareRevision:** Recupera la versión del firmware de un dispositivo.
  - **cmtGetLastHwError:** Devuelve el código de error del problema que ha pasado en el hardware.
  - **cmtGetProductCode:** Recupera el código del producto de un dispositivo.
  - **cmtGotoConfig:** Cambiar al modo de configuración.
  - **cmtGotoMeasurement:** Cambiar al modo de medición.
  - **cmtReset:** Resetea el dispositivo.
  - **cmtResetOrientation:** Realiza un restablecimiento de la orientación en un dispositivo. Se debe de especificar el ID del dispositivo.
  - **cmtRestoreFactoryDefaults:** Restaura completamente el dispositivo a la configuración por defecto de fábrica.
  - **cmtSendCustomMessage:** Envía un mensaje personalizado al dispositivo seleccionado.
  - **cmtSetSoftwareCalibration:** Activa o desactiva el software de la calibración.
  - **cmtSetSoftwareXkf3Filtering:** Activa o desactiva el software del XKF-3.

2.1.4.3 Estructura de los posibles mensajes en el campo MID

En este apartado se ofrece una referencia de todos los mensajes válidos que se pueden poner, con sus identificadores, en el campo MID en la estructura del mensaje.

Mensajes de WakeUp y estado

Mensaje	MID	Dirección	Descripción
WakeUp	62(0x3E)	Hacia controlador	El dispositivo envía este mensaje al encender o reiniciar
WakeUpAck	63(0x3F)	Hacia MT	Si se recibe dentro de 500 ms después del WakeUp el dispositivo entra en el estado de configuración, si no se recibe, entra en el estado de medición
GoToConfig	48(0x30)	Hacia MT	El dispositivo entra en el estado de configuración
GoToConfigAck	49(0x31)	Hacia controlador	El dispositivo reconoce el mensaje GoToConfig
GoToMeasurement	16(0x10)	Hacia MT	El dispositivo entra en el estado de medición

GoToMeasurementAck	17(0x11)	Hacia controlador	El dispositivo reconoce el mensaje GoToMeasurement
Reset	64(0x40)	Hacia MT	Reset del dispositivo
ResetAck	65(0x41)	Hacia controlador	Acuse de recibo del mensaje de Reset

Mensajes de información

Mensaje	**MID**	**Dirección**	**Descripción**
ReqDID	0(0x00)	Hacia MT	Solicitud del controlador del dispositivo del ID
DeviceID	1(0x01)	Hacia controlador	El dispositivo reconoce la solicitud mediante el envío de su ID
ReqProductCode	28(0x1C)	Hacia MT	Petición del controlador del código de producto del dispositivo
ProductCode	29(0x1D)	Hacia controlador	Dispositivo reconoce la solicitud mediante el envío del código de producto
ReqFWRev	18(0x12)	Hacia MT	El controlador pide la revisión de firmware del dispositivo
FirmwareRev	19(0x13)	Hacia controlador	Dispositivo admite el envío de su solicitud de revisión de firmware
ReqDataLength	10(0x0A)	Hacia MT	Solicita el número de bytes de datos en el mensaje MTData
DataLength	11(0x0B)	Hacia controlador	Contiene el número de bytes de datos del mensaje MTData
Error	66(0x42)	Hacia controlador	Mensaje de error

Mensajes específicos del dispositivo

Mensaje	**MID**	**Dirección**	**Descripción**
Reqbaudrate	24(0x18)	Hacia MT	Solicita la actual velocidad de transmisión de la comunicación en serie
ReqbaudrateAck	25(0x19)	Hacia controlador	Devuelve la velocidad de transmisión de la comunicación en serie
SetBaudrate	24(0x18)	Hacia MT	El controlador establece la velocidad de trasmisión (en baudios) de la comunicación serie
SetBaudrateAck	25(0x19)	Hacia controlador	El dispositivo reconoce el mensaje SetBaudrate

ReqErrorMode	218(0xDA)	Hacia MT	Solicitud de modo de error
ReqErrorModeAck	219(0xDB)	Hacia controlador	El dispositivo recupera el modo de error
SetErrorMode	218(0xDA)	Hacia MT	El controlador establece el modo de error
SetErrorModeAck	219(0xDB)	Hacia controlador	El dispositivo reconoce el mensaje SetErrorMode
ReqLocationID	132(0x84)	Hacia MT	Solicitud de la ubicación del ID
ReqLocationIDAck	133(0x85)	Hacia controlador	El dispositivo recupera la ubicación del ID
RestoreFactoryDef	132(0x84)	Hacia MT	El controlador establece la ubicación del ID
RestoreFactoryDefAck	133(0x85)	Hacia controlador	El dispositivo reconoce el mensaje SetLocationID
ReqTransmitDelay	220(0xDD)	Hacia MT	Solicita el valor de retraso, lo que equivale a el tiempo mínimo entre la recepción y el último byte de inicio de la transmisión
ReqTransmitDelayAck	221(0xDC)	Hacia controlador	El dispositivo recupera el valor de retardo
SetTransmitDelay	220(0xDD)	Hacia MT	El controlador establece el valor de retardo
SetTransmitDelayAck	221(0xDC)	Hacia controlador	El dispositivo reconoce el mensaje SetTransmitDelay
StoreXkfState	138(0x8A)	Hacia MT	Almacena el estado de lXKF en la memoria no volátil del dispositivo

Mensajes de sincronismo

Mensaje	**MID**	**Dirección**	**Descripción**
ReqSyncInSettings	214(0xD6)	Hacia MT	Solicita un ajuste de SyncIn del dispositivo. Ejemplo de modo SyncIn, omitir el factor o compensar.
ReqSyncInSettingsAck	215(0xD7)	Hacia controlador	El dispositivo devuelve el ajuste del SyncIn
SetSyncInSettings	214(0xD6)	Hacia MT	El controlador establece el ajuste del SyncIn
SetSyncInSettingsAck	215(0xD7)	Hacia controlador	El dispositivo responde con el mensaje SetSyncInSettings
ReqSyncOutSettings	216(0xD8)	Hacia MT	Solicita un ajuste de SyncOut del dispositivo. Ejemplo de modo SyncOut, omitir factor, compensar o ajustar el ancho

			de pulso
ReqSyncOutSettingsAck	217(0xD9)	Hacia controlador	El dispositivo devuelve el ajuste del SyncOut
SetSyncOutSettings	216(0xD8)	Hacia MT	El controlador establece el ajuste del SyncOut
SetSyncOutSettingsAck	217(0xD9)	Hacia controlador	El dispositivo responde con el mensaje SetSyncOutSettings

Mensajes de configuración

Mensaje	**MID**	**Dirección**	**Descripción**
ReqConfiguration	12(0x0C)	Hacia MT	Solicita a la configuración del dispositivo.
Configuration	13(0x0D)	Hacia controlador	Contiene la configuración del dispositivo
ReqPeriod	4(0x04)	Hacia MT	Solicita el periodo de muestreo
ReqPeriodAck	5(0x05)	Hacia controlador	El dispositivo recupera el período de muestreo.
SetPeriod	4(0x04)	Hacia MT	El controlador establece el periodo de muestreo (10 – 500Hz)
SetPeriodAck	5(0x05)	Hacia controlador	El dispositivo responde con el mensaje SetPeriod.
ReqOutputSkipFactor	212(0xD4)	Hacia MT	Solicitud del número de veces quela salida de datos se pasa por alto antes de enviar un mensaje MTData.
ReqOutputSkipFactorAck	213(0xD5)	Hacia controlador	El dispositivo devuelve OutputSkipFactor.
SetOutputSkipFactor	212(0xD4)	Hacia MT	El controlador envía el OutputSkipFactor
SetOutputSkipFactorAck	213(0xD5)	Hacia controlador	El dispositivo responde con el mensaje SetOutputSkipFactor
ReqObjectAlignment	224(0xE0)	Hacia MT	Solicita el objeto de la matriz alineación
ReqObjectAlignmentAck	225(0xE1)	Hacia controlador	El dispositivo devuelve el objeto de la matriz alineación
SetObjectAlignment	224(0xE0)	Hacia MT	Establece el objeto de la matriz alineación.
SetObjectAlignmentAck	225(0xE1)	Hacia controlador	El dispositivo responde con el mensaje SetObjectAlignment
ReqOutputMode	208(0xD0)	Hacia MT	Solicita el modo de salida
ReqOutputModeAck	209(0xD1)	Hacia controlador	El dispositivo devuelve el modo de salida
SetOutputMode	208(0xD0)	Hacia MT	El controlador envía el modo de salida

SetOutputModeAck	209(0xD1)	Hacia controlador	El dispositivo responde con el mensaje SetOutputMode
ReqOutputSettings	210(0xD2)	Hacia MT	Solicita el ajuste de salida.
ReqOutputSettingsAck	211(0xD3)	Hacia controlador	El dispositivo devuelve el ajuste de salida
SetOutputSettings	210(0xD2)	Hacia MT	El controlador envía el ajuste de salida
SetOutputSettingsAck	211(0xD3)	Hacia controlador	El dispositivo responde con el mensaje SetOutputSettings

Mensajes relacionados con datos

Mensaje	**MID**	**Dirección**	**Descripción**
ReqData	52(0x34)	Hacia MT	El controlador pide al dispositivo enviar un mensaje MTData
MTData	50(0x32)	Hacia controlador	Mensaje con los datos en bruto no calibradas, los datos de calibrado o datos de orientación

Filtro de mensajes XKF

Mensaje	**MID**	**Dirección**	**Descripción**
ReqHeading	130(0x82)	Hacia MT	Solicita la configuración departida
ReqHeadingAck	131(0x83)	Hacia controlador	El dispositivo devuelve la configuración de partida
SetHeading	130(0x82)	Hacia MT	El controlador establece el modo de salida
SetHeadingAck	131(0x83)	Hacia controlador	El dispositivo responde con el mensaje SetHeading
ResetOrientation	164(0xA4)	Hacia MT	Reset de la orientación
ResetOrientationAck	165(0xA5)	Hacia controlador	El dispositivo responde con el mensaje ResetOrientation
ReqUTCTime	96(0x60)	Hacia MT	Solicita el tiempo UTC
UTCTime	97(0x61)	Hacia controlador	El dispositivo devuelve el tiempo UTC
ReqAvailableScenarios	98(0x62)	Hacia MT	Solicita los escenarios disponibles
AvailableScenarios	99(0x63)	Hacia controlador	El dispositivo devuelve los escenarios disponibles
ReqCurrentScenario	100(0x64)	Hacia MT	Solicita los actuales escenarios utilizados
ReqCurrentScenarioAck	101(0x65)	Hacia controlador	El dispositivo devuelve los escenarios actuales

SetCurrentScenario	100(0x64)	Hacia MT	El controlador establece los escenarios actuales
SetCurrentScenarioAck	101(0x65)	Hacia controlador	El dispositivo responde con el mensaje SetCurrentScenario
ReqGravityMagnitude	102(0x66)	Hacia MT	Solicita la magnitud actual de la gravedad utilizada
ReqGravityMagnitudeAck	103(0x67)	Hacia controlador	El dispositivo recupera la magnitud actual de la gravedad.
SetGravityMagnitude	102(0x66)	Hacia MT	El controlador establece la magnitud de la gravedad.
SetGravityMagnitudeAck	103(0x67)	Hacia controlador	El dispositivo responde con el mensaje SetGravityMagnitude
ReqMagneticDeclination	106(0x6A)	Hacia MT	Solicita la actual declinación magnética utilizada
ReqMagneticDeclinationAck	107(0x6B)	Hacia controlador	El dispositivo recupera la declinación magnética actual
SetMagneticDeclination	106(0x6A)	Hacia MT	El controlador establece la declinación magnética actual
SetMagneticDeclinationAck	107(0x6B)	Hacia controlador	El dispositivo responde con el mensaje SetMagneticDeclination
ReqProcessingFlags	32(0x20)	Hacia MT	Solicita el filtro de procesamiento de los flags
ReqProcessingFlagsAck	33(0x21)	Hacia controlador	El dispositivo devuelve el procesamiento de los flags
SetProcessingFlags	32(0x20)	Hacia MT	El controlador establece el procesamiento de los flags
SetProcessingFlagsAck	33(0x21)	Hacia controlador	El dispositivo responde con el mensaje SetProcessingFlags
SetNoRotation	34(0x22)	Hacia MT	IniciaXKF3"no hay rotación" procedimiento de actualización
SetNoRotationAck	35(0x23)	Hacia controlador	El dispositivo responde con el mensaje SetNoRotation

2.1.5 Estructura del Hardware del sensor MTi

En este apartado se hará referencia a las especificaciones físicas del sensor y los sensores con los que cuenta el MTi que son acelerómetros, giroscopios y magnetómetros. También se describirán los componentes del sensor como son su fuente de alimentación, el cable serial USB, las características de su conector y el tamaño del sensor.

2.1.5.1 Visión general de la física del sensor

En la Tabla 6 se muestran como son los tipos de sensores con los que cuenta el MT. Además, el MTi tiene varios sensores de temperatura a bordo para permitir la compensación por dependencia de la temperatura de los diferentes sensores.

Tabla 6: Tipos de sensores

Tabla de los sensores MTi	
Acelerómetros	MEMS de estado sólido, lectura capacitiva
Sensor de velocidad de giro (giroscópio)	MEMS de estado sólido, monolítico, haz de estructura, lectura capacitiva
Magnetómetro	Película delgada magnetoresistiva

Las características del sensor se muestran en la Tabla 7 donde se describe el tipo de comunicación con el que cuenta el sensor, voltaje de funcionamiento, potencia consumida, temperatura que soporta, tamaño y peso.

Tabla 7: Características del sensor MTi

MTi – 28A	
Interfaz de comunicación	Serial digital (RS - 232)
Interfaces adicionales	SyncIn SyncOut Analog in
Voltaje	4.5 a 30 V
Potencia consumida	350 mW
Temperatura Rango de Operación	-20^0C a 55^0C
Dimensiones	58 x 59 x 22 (W x L x H)
Peso	50 g

2.1.5.2 Componentes

Fuente de alimentación

La fuente de alimentación nominal del MTi es de 5 V en DC

La tensión mínima de funcionamiento de es > 4,5 V y el máximo absoluto es de < 30 V.

- El sensor trabaja con una fuente de alimentación entre 4.5-30 V. Utilizar sólo fuentes de alimentación SELV (‘*Separated of safety Extra Low Voltage’*) que son a prueba de cortocircuitos (doble aislamiento).
- El consumo de energía promedio de operación es 350mW (~ 70 mA a 5 V) para el MTi. El consumo medio de energía puede variar ligeramente con el modo de uso. Tenga en cuenta que la eficiencia de la etapa de entrada de energía se reducirá con el aumento de la tensión de alimentación. Entre 5-6 V de DC, la eficacia es óptima, a 30V de DC, la eficiencia es de alrededor de 75%.
- El pico de corriente en el encendido pueden ser de hasta 200 mA.
- Cuando se opera a temperatura ambiente la temperatura dentro del sensor variará entre 33-40 ° C en condiciones normales.

Cable serie USB y de alimentación

En el Kit de Desarrollo MTi se suministra el cable de datos serie USB que sirve además de alimentación. Este cable es compatible con USB 1.1 y superior. Hay que asegurarse de que la salida USB del PC está cualificada para entregar 100 mA o más. A continuación en la Figura 17 se muestra dicho cable.

Figura 17: Cable serie USB

El USB de datos y cable de alimentación proporcionan un fácil acceso a las clavijas individuales del MTi. Dentro de la carcasa hay un conector libre que puede ser usado, por ejemplo, para propósitos de sincronización. La Figura 18 muestra la ubicación de dicho conector.

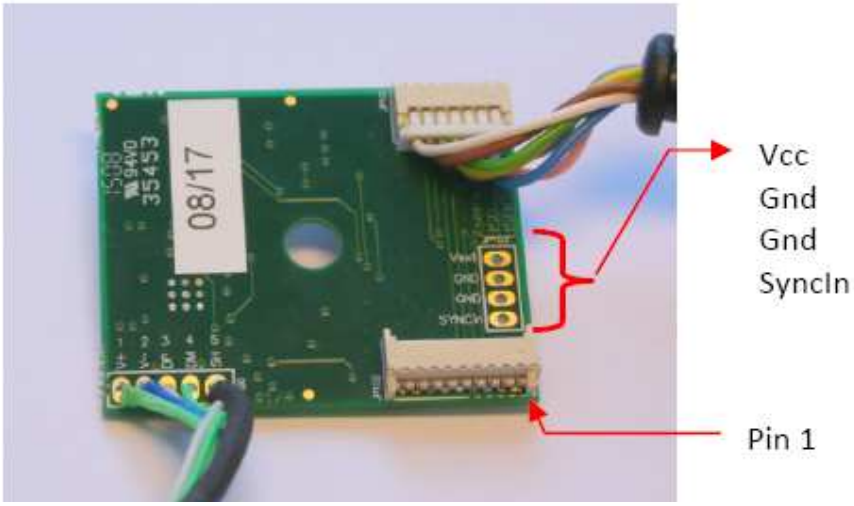

Figura 18: Interior del sensor

El MTi está diseñado para ser utilizado con la alimentación proporcionada por Xsens (integrado en el RS-232 a USB cable). Es posible utilizar otras fuentes de alimentación, sin embargo esto debe hacerse con cuidado. Por razones de seguridad cualquier fuente de alimentación que se utilice con el dispositivo debe cumplir con la directiva de compatibilidad electromagnética.

Los pines y definiciones de los colores del cable del MTi son distintos dependiendo del MTi que se utilice, en nuestro caso MTi-28A (MTi RS-232, versión estándar).

El conector hembra que está situado en el MTi es un ODU serie L de 7 como el que se puede observar en la Figura 19.

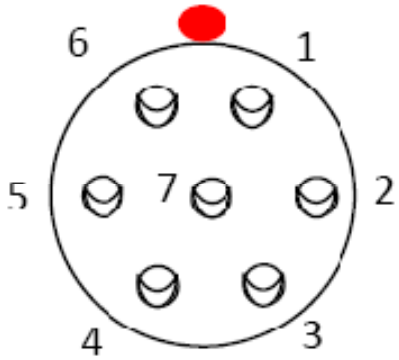

Figura 19: Clavija hembra del sensor

Las definiciones de los 7 pines del conector hembra se muestran a continuación en la Tabla 8.

Tabla 8: Conexionado de los pines (hembra)

Molex pin	MTi RS-232
Pin 1	VCC
Pin 2	GND
Pin 3	Analog IN
Pin 4	TX (sensor)
Pin 5	RX (sensor)
Pin 6	SyncOut
Pin 7	SyncIn

El conector macho del MTi es un ODU serie L de 7 pines que viene en el convertidor RS232-USB. A continuación, en la Figura 20, se muestra la clavija y la numeración de los pines.

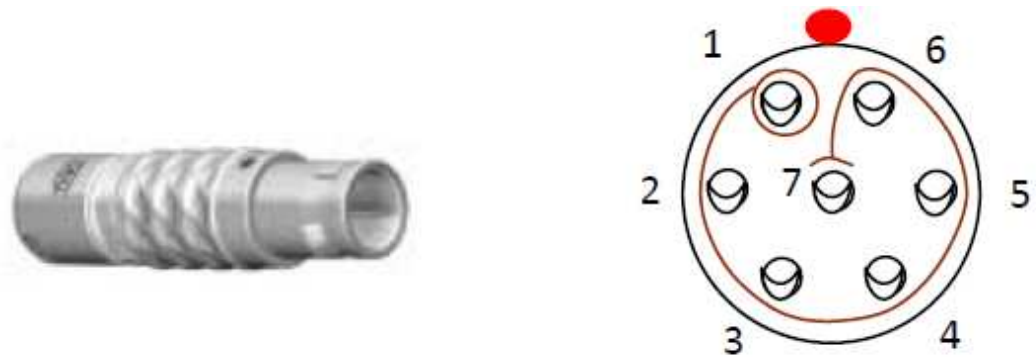

Figura 20: Clavija macho del cable del sensor

La definición de los pines del conector macho del MTi se muestra en la Tabla 9 y el color del cableado de éste se puede observar en la Tabla 10, las cuales se muestran a continuación.

Tabla 9: Conexionado de los pines (macho)

Signal	ODU pin
VCC	Pin 1
GND	Pin 2
Y / A	Pin 3
Z / B	Pin 4
Reserved	Pin 5
SyncOut	Pin 6
SyncIn	Pin 7

Tabla 10: Color de los cables de los pines

Vector	Cable unitronic	Cable elitronic
Pin 1	Amarillo	Blanco
Pin 2	Amarillo - verde	Marrón
Pin 3	Negro	Verde
Pin 4	Beige	Amarillo
Pin 5	Marrón	Gris
Pin 6	Verde	Rosa
Pin 7	Azul	Azul

Carcasa

Las piezas de plástico de la carcasa son de poliamida (PA6.6). La placa inferior MTi está hecha de aluminio anodizado (6082). La carcasa es a prueba de polvo, pero no a prueba de agua. La toma del conector de MTi y el ensamblaje de la carcasa es de goma en forma de anillo.

La carcasa MTi está diseñada para soportar el uso en aplicaciones donde el polvo y las salpicaduras de agua ocasional se puedan esperar. Los test de Xsens han confirmado que la carcasa y el conector pueden soportar circunstancias ambientales temporales equivalentes a las de protección IP 66 (selladas contra el polvo, protección contra el chorro de agua de gran alcance). Tenga en cuenta que el conector de la carcasa MTi es resistente al agua, pero el conector que se suministra no es a prueba de agua.

El material plástico utilizado para MTi tiene la clasificación UL94 V2.

Las dimensiones del MTi se muestran a en la Figura 21 para más información consultar el anexo A.3 DIMENSIONES SENSOR MTi DE XSENS

Figura 21: Dimensiones del sensor MTi

2.1.5.3 Localización del origen físico

El MTi es ante todo, un sensor de orientación y como tal, no es importante donde tiene su origen interno, es decir, la orientación es la misma para todas las posiciones del MT, ya que puede ser considerado como un cuerpo rígido. Sin embargo, para aplicaciones donde se miden las aceleraciones es importante conocer el verdadero origen del MT porque se define por la ubicación física del acelerómetro.

La Figura 22 muestra cómo se puede encontrar el vector de traslación entre el origen del MT (**0**) y un cierto punto en el dispositivo externo que sea el más conveniente, este punto puede ser **0'** (un agujero del tornillo) u **O''** (la intersección entre las partes del MTi en el exterior de la carcasa).

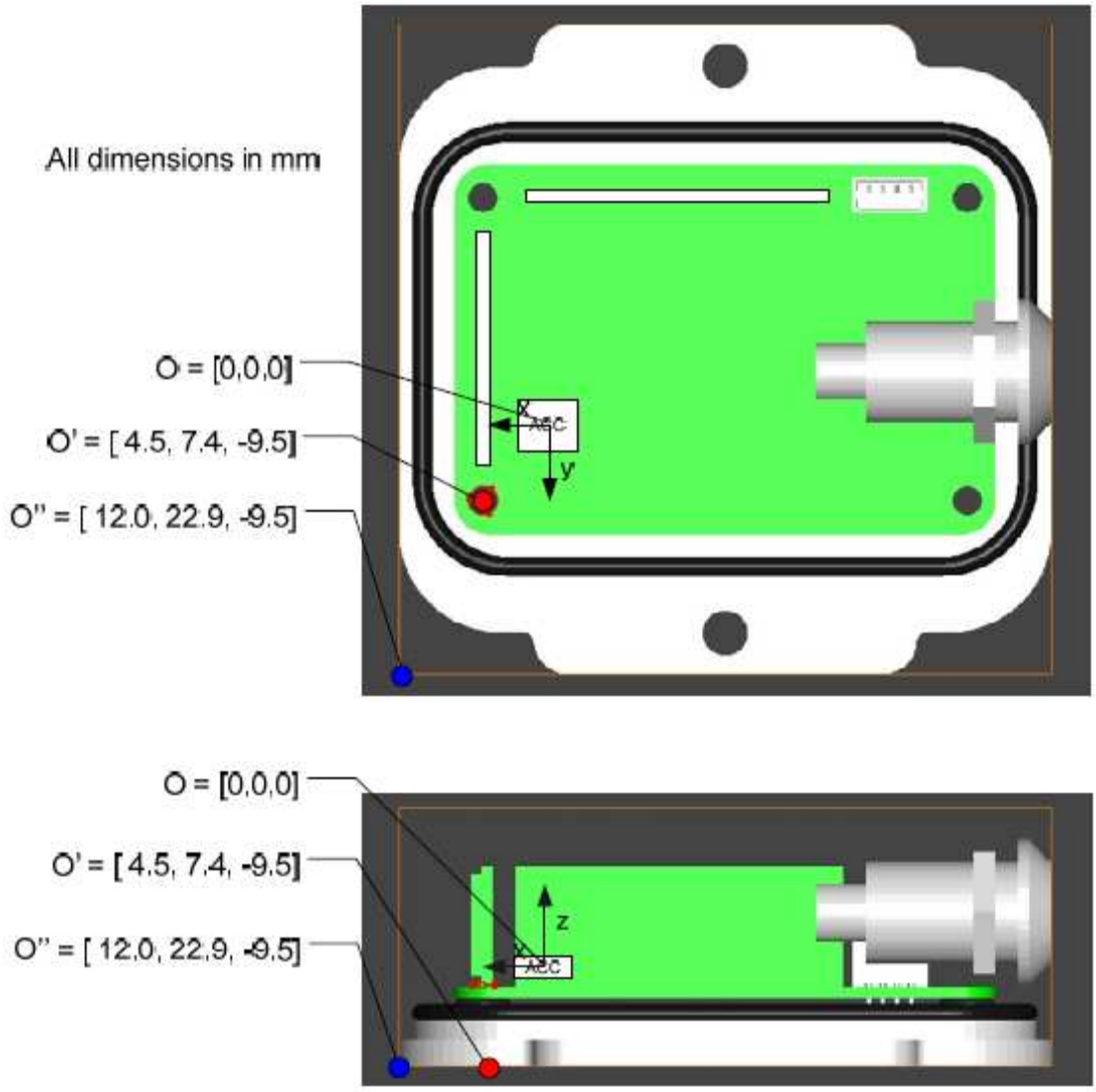

Figura 22: Posición del eje de coordenadas

2.1.6 Consideraciones

Hay tres consideraciones que hay que tener en cuenta a la hora de colocar el MTi en un objeto o cuerpo y son las aceleraciones transitorias, las vibraciones y los materiales magnéticos.

- **Aceleraciones transitorias**

  Los acelerómetros lineales 3D del MTi se utilizan principalmente para estimar la dirección de la gravedad, para obtener una referencia del sensor (‘*pitch*’ / ‘*roll*’). Durante largos períodos de tiempo (más de unos pocos segundos) de aceleraciones transitorias (es decir, derivada segunda de la posición) la

observación de la gravedad no se puede hacer. Los algoritmos de fusión de sensores XKF toman en cuenta estos efectos, pero sin embargo es imposible estimar la verdadera vertical sin información adicional.

El impacto de aceleraciones transitorias se puede minimizar si se toma en cuenta algunas consideraciones durante la colocación del dispositivo.

Si desea utilizar el MTi para medir la dinámica de un vehículo en movimiento, embarcaciones, lo mejor es colocar el dispositivo de medición en una posición donde se espera que haya menos aceleraciones transitorias. Esto es típicamente cerca del centro de gravedad del vehículo / embarcación, ya que cualquier rotación alrededor del centro de gravedad se traduce en aceleraciones centrípetas en cualquier punto fuera del punto de rotación, que es generalmente cerca del centro de gravedad. La aceleración del vehículo en su conjunto no puede tenerse en cuenta.

- **Vibraciones**

Para un mejor funcionamiento del MTi, deberá estar mecánicamente aislado de las vibraciones tanto como sea posible. Las vibraciones se miden directamente por los acelerómetros. Esto no es necesariamente un problema, pero dos condiciones pueden hacer que las lecturas de los acelerómetros sean inválidas:

1. La magnitud de la vibración es mayor que el rango del acelerómetro. Esto hará que el acelerómetro se sature, lo que puede ser observado como un '*drift*' en el nivel cero del acelerómetro. El mismo se mostrará en las estimaciones de la orientación 3D como un error de '*pitch*' / '*roll*'.

2. La frecuencia de la vibración es mayor que el ancho de banda del acelerómetro. En teoría, estas vibraciones son rechazadas, pero en la práctica todavía puede dar lugar a '*aliasing*' (efecto que causa que señales continuas distintas se tornen indistinguibles cuando se muestrean digitalmente), sobre todo si está cerca del límite del ancho de banda. Esto se puede observar como una oscilación de baja frecuencia. Además, las vibraciones de alta frecuencia a menudo tienden a tener grandes amplitudes de aceleración.

- **Los materiales magnéticos y los imanes**

Cuando un MTi se coloca cerca o sobre un objeto que contiene materiales ferromagnéticos, o que es magnético por sí mismo, la medida del campo magnético es distorsionada y provoca un error en la medida de orientación. El

campo magnético terrestre se ve alterado por los materiales ferromagnéticos, los imanes permanentes o corrientes muy fuertes (varios amperios). En la práctica, la distancia al objeto y la cantidad de material ferromagnético determina el grado de perturbación. Los errores en la orientación debido a estas distorsiones pueden ser muy grandes, ya que el campo magnético terrestre es muy débil en comparación con la magnitud de las muchas fuentes de distorsión.

Sea o no un objeto ferromagnético, preferentemente, deberá ser revisado por el uso de los magnetómetros del MTi. También se puede comprobar con un pequeño imán, pero tenga cuidado, usted puede fácilmente magnetizar materiales ferromagnéticos duros, provocando errores aún más grandes. Esto es a menudo el caso con, por ejemplo, aceros inoxidables, que normalmente no son magnéticos, pero se encuentran magnetizados, pues puede ser posible "la desmagnetización" del objeto.

En la mayoría de los casos en que la alteración del campo magnético causado por la colocación del MTi en un objeto ferromagnético se puede corregir para el uso de un procedimiento de calibración. El procedimiento de calibración se puede ejecutar en pocos minutos y produce un nuevo conjunto de parámetros de calibración que se puede escribir en la memoria no volátil del MTi.

Este procedimiento de calibración se lleva a cabo en el módulo de software '*Magnetic Field Mapper*' que viene con el SDK. El método utilizado en este software es único en el sentido de que permite a un usuario elegir la secuencia de medición (con ciertas limitaciones), y permite el mapeo 3D.

Capítulo 3: HERRAMIENTA SOFTWARE

3.1 INTRODUCCIÓN A LA INTERFAZ GRÁFICA EN MATLAB

MATLAB (abreviatura de '*MATrix LABoratory*', "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio [3].

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI).

En este proyecto utilizaremos Matlab para hacer la interfaz gráfica de configuración con el usuario, además de esta interfaz, se ha realizado un fichero mex en Matlab que sirve de unión entre el programa principal.cpp y la interfaz gráfica, ya que comparte los datos a través de una memoria compartida, con el programa en C++ que es el encargado de la comunicación con el sensor MT,

Una aplicación GUIDE consta de dos archivos: *.m* y *.fig*. El archivo *.m* es el ejecutable y el *.fig* la parte gráfica.

3.2 REALIZACIÓN DE UNA GUIDE

Para crear una interfaz, lo primero que hay que hacer es abrir la aplicación, se puede escribir en la consola de Matlab directamente la palabra "guide" o bien se puede pulsar el icono que muestra la Figura 23 [4].

Figura 23: Icono GUIDE

Al pulsar el botón GUIDE se representará la ventana que aparece en la Figura 24 donde se pulsará la opción '*Blank Gui'*, para crear una interfaz gráfica de usuario en blanco, nos presenta un formulario nuevo, en el cual podemos diseñar nuestro programa.

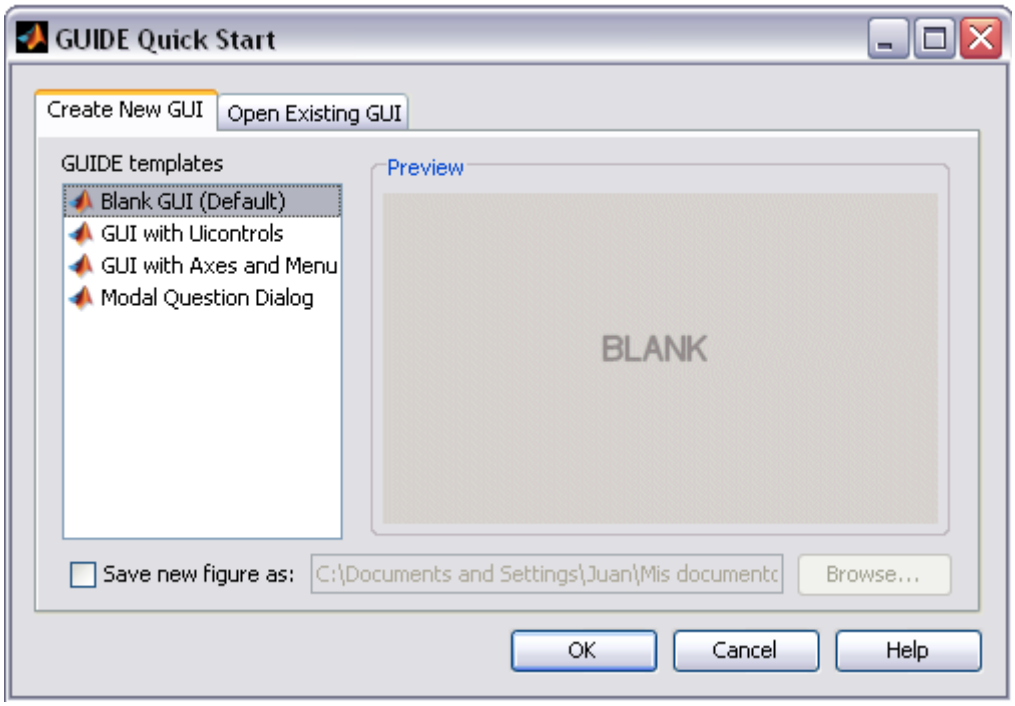

Figura 24: Ventana de inicio de GUIDE

Una vez que tenemos el formulario nuevo mostrado en la Figura 25 se pueden diferenciar tres zonas:

1. **Área de diseño:** donde se compone la interfaz.
2. **Paleta de componentes:** que expone las diferentes opciones que se pueden realizar.
3. **Herramientas:** para realizar ajustes y cambios en los componentes.

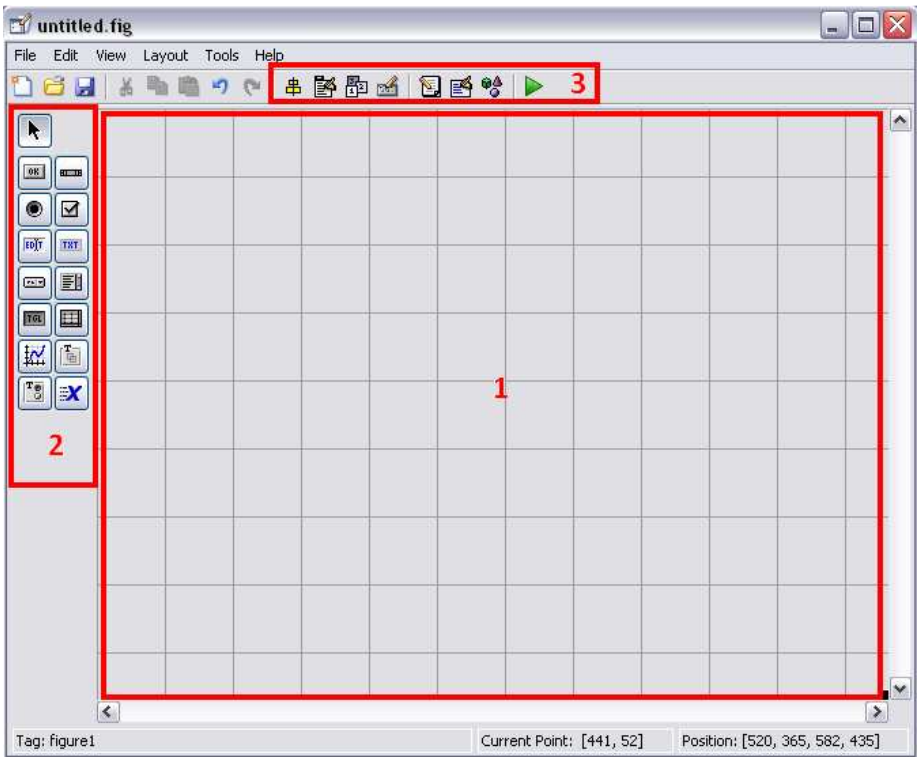

Figura 25: Entorno de diseño GUI

En la Tabla 11 se muestran las siguientes herramientas que son necesarias para la creación de la GUIDE.

Tabla 11: Herramientas

	Alinear objetos
	Editor de menú
	Editor de orden de etiqueta
	Editor del M file
	Propiedades
	Navegador
	Grabar y ejecutar

La descripción de la paleta de componentes se muestra en la Tabla 12.

Tabla 12: Descripción tableta de componentes.

Control	Valor de estilo	Descripción
Check box	'checkbox'	Indica el estado de una opción o atributo
Editable Text	'edit'	Caja para editar texto
Pop-up menu	'popupmenu'	Provee una lista de opciones
List Box	'listbox'	Muestra una lista deslizable
Push Button	'pushbutton'	Invoca un evento inmediatamente
Radio Button	'radio'	Indica una opción que puede ser seleccionada
Toggle Button	'togglebutton'	Solo dos estados, "on" o "off"
Slider	'slider'	Usado para representar un rango de valores
Static Text	'text'	Muestra un string de texto en una caja
Panel button		Agrupa botones como un grupo
Button Group		Permite exclusividad de selección con los radio button
Axes		Realiza Graficas

Cada uno de los elementos de GUI, tiene un conjunto de opciones que podemos acceder con clic derecho. Esta opción se observa en la Figura 26.

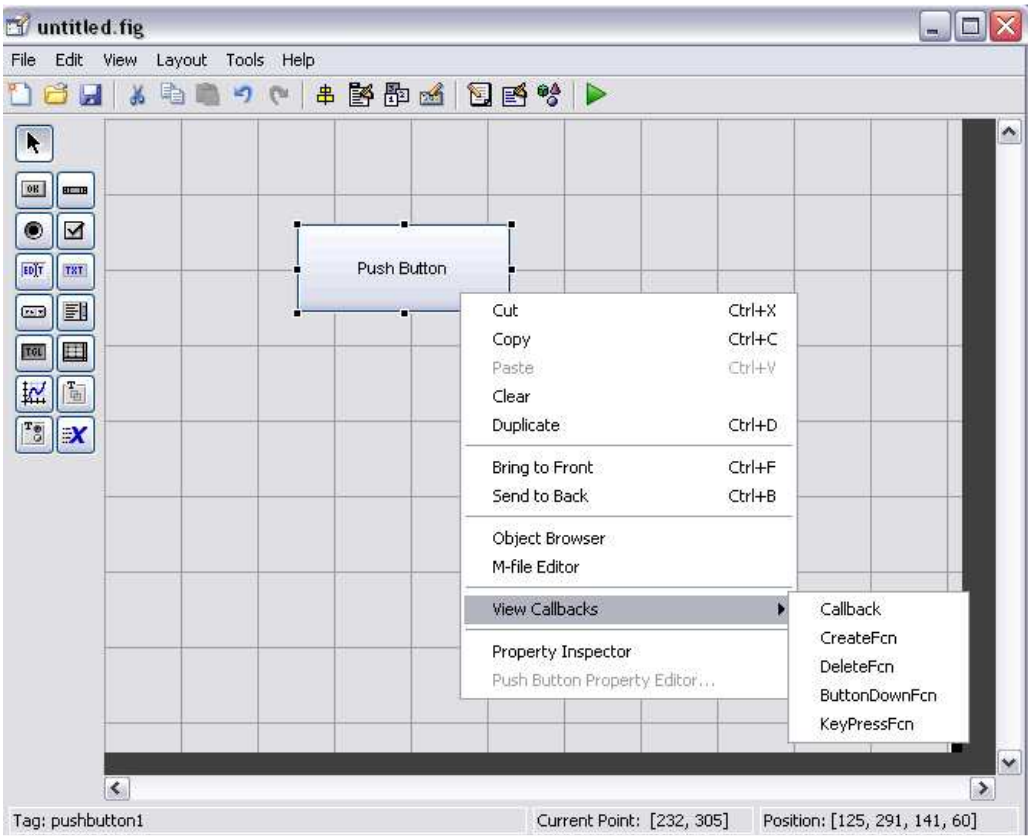

Figura 26: Opciones del componente

La opción '*Property Inspector*' es la mostrada en la Figura 27, la cual nos permite personalizar cada elemento.

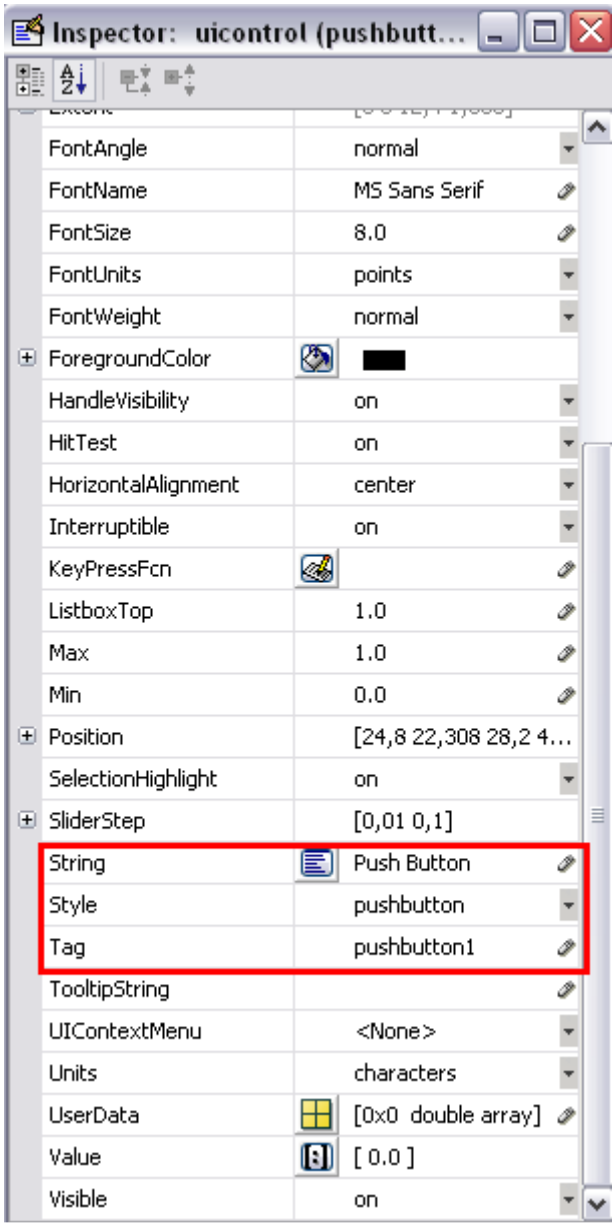

Figura 27: Entorno Property Inspector

Las propiedades más importantes son '*String*' que sirve para modificar el nombre que se mostrará en el pulsador, '*Style*' que describe el componente, se puede modificar pulsando en la flecha y por último, en el campo '*tag*' podemos cambiar el nombre con el que aparecerá la subfunción del '*pushbutton*' en el m-file,

Una de las opciones más importantes que aparecen al hacer clic derecho en el elemento ubicado en el área de diseño, es '*View Callbacks*', la cual, al ejecutarla, abre el archivo .m asociado a nuestro diseño y nos posiciona en la parte del programa que corresponde a la subrutina que se ejecutará cuando se realice una determinada acción sobre el elemento que estamos editando.

Por ejemplo, al ejecutar *View Callbacks>>Callbacks* en el *Push Button*, nos ubicaremos en la parte del programa donde escribiremos la función correspondiente a la acción del pulsar el *Push Button*:

```
function pushbutton1_Callback(hObject, eventdata, handles)

%hObject handle to pushbutton1 (see GCBO)
%eventdata reserved - to be defined in a future version of MATLAB
%handles structure with handles and user data (see GUIDATA)
```

3.3 FICHEROS MEX

Es posible llamar desde Matlab a funciones programadas en C como si fueran funciones propias de Matlab. De este modo, una función *.m de Matlab puede ser sustituida por una función programada en C que se llama exactamente de la misma forma. Para que esto sea posible las funciones programadas en C han de cumplir una serie de requisitos que se explican más adelante. Estas funciones se compilan y generan librerías compartidas que son las denominadas funciones MEX.

Las funciones MEX tienen una extensión diferente en función de los sistemas operativos en que hayan sido generadas, en nuestro caso, la extensión es .mexglx, ya que utilizamos Linux.

3.3.1 Creación de ficheros MEX

El código fuente de un fichero MEX programado en C tiene dos partes. La primera parte contiene el código de la función C que se quiere implementar como fichero MEX. La segunda parte es la función mexFunction que hace de interface entre C y Matlab.

La función mexFunction tiene cuatro argumentos: prhs, nrhs, plhs y nlhs. Estos argumentos tienen los siguientes significados:

1. **prhs** es un vector de punteros a los valores de los argumentos de entrada (‘*right hand side arguments*’) que se van a pasar a la función C.

2. **nhrs** es el número de argumentos de entrada de la función.

3. **plhs** y **nlhs** son análogos pero referidos a los argumentos de salida (‘*left hand side arguments’*).

Antes de seguir adelante con esta explicación es conveniente decir algo sobre los mxArrays, que son los únicos objetos con los que trabaja Matlab. Todos los tipos de variables de Matlab (escalares, vectores, matrices, cadenas de caracteres, estructuras, vectores de celdas, etc.) son mxArrays. Para cada mxArray, Matlab almacena el tipo, las dimensiones, los datos, si es real o complejo (para datos numéricos), el número de campos y sus nombres para las estructuras, etc. Matlab dispone de un gran número de funciones C para trabajar con mxArrays, que pueden encontrarse buscando ‘*MX Array Manipulation (C)*’ en el ‘*Help*’.

A continuación, se describe con más generalidad la creación de ficheros MEX. En la Figura 28, inspirada en el ‘*Help*’ de MATLAB, se ha querido mostrar cómo se realiza la comunicación entre C y MATLAB, cómo llegan los datos al fichero MEX, qué es lo que se hace con estos datos en mex-Function y cómo se devuelven finalmente los resultados a MATLAB.

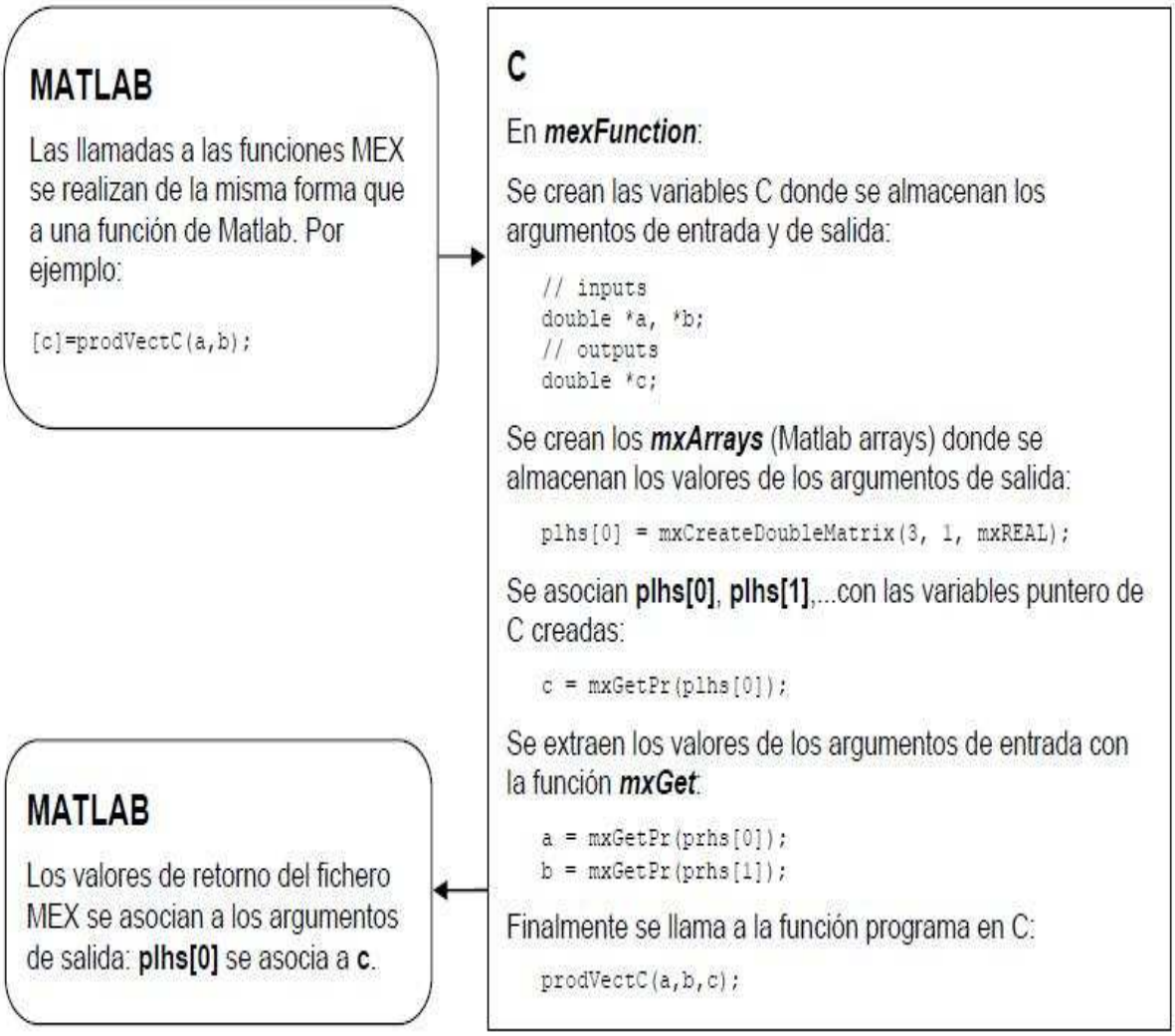

Figura 28: Esquema general de la creación de una funcion MEX

Los ficheros MEX deben incluir la librería "mex.h" donde está declarada la función mexFunction, cuya cabecera es la siguiente:

*void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])*

Capítulo 4: DESARROLLO DE LA APLICACIÓN

4.1 INTRODUCCIÓN

Este proyecto consiste en realizar una comunicación básica entre el sensor y el usuario para que éste pueda configurarlo y recibir los datos requeridos, para ello se ha desarrollado una serie de programas que se detallan a continuación.

El primer programa es el llamado módulo de comunicación, que está programado en lenguaje C++, es el encargado de comunicarse con el sensor y mandar los datos pedidos. La otra parte es un programa en Matlab que a su vez engloba la interfaz gráfica donde se mostrarán los datos y el programa union.mexglx, el cual, como su nombre expresa, sirve como nexo de unión entre la interfaz gráfica y el módulo de comunicación a través de una zona de memoria compartida.

El esquema de la comunicación entre el usuario y el sensor MTi se puede observar en la Figura 29.

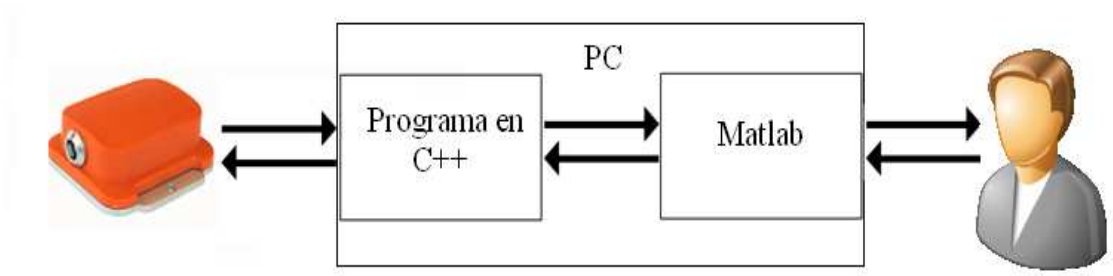

Figura 29: Esquemático de comunicación

Y el esquema de comunicación del módulo en Matlab es el mostrado en la Figura 30.

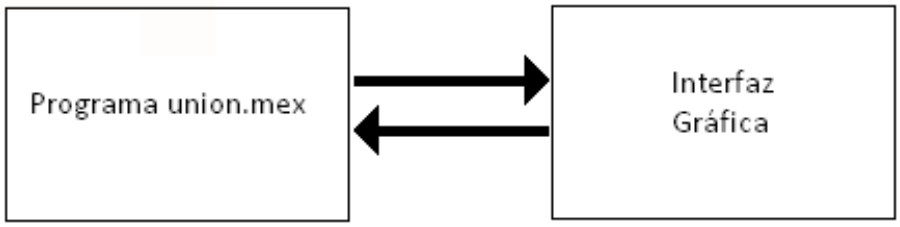

Figura 30: Esquemático de comunicación en Matlab

El usuario podrá configurar el sensor a través de la aplicación en Matlab, que ésta a su vez, se comunicará con el programa en C++ (a través de memorias compartidas) que se comunicará con el sensor y obtendrá los datos pedidos para a continuación, hacer el camino inverso para mostrar dichos datos en la interfaz de Matlab.

A lo largo de este capítulo se detallara todo este proceso de comunicación desde que los datos salen del sensor y llegan al usuario como viceversa.

4.2 PROGRAMACIÓN DEL MÓDULO DE COMUNICACIÓN

El programa encargado de la comunicación con el sensor se llama principal.cpp, está realizado en lenguaje C++, bajo el sistema operativo Linux y su desarrollo se describe en el diagrama de flujo que se observa en la Figura 31.

Para una mejor visualización del código se ha realizado una única librería llamada funcion.h donde se engloban todas las librerías típicas de programación en Linux, las librerías especiales del sensor y la librería memoria.h que será la encargada de la memoria compartida, también se han definido las funciones que utilizaremos, una serie de variables globales y el tipo de estructuras donde se guardarán los datos.

Este programa se compila por medio un archivo Makefile, el cual, hace referencia a las librerías internas del sensor para poder crear el archivo ejecutable .cpp

El programa generado “principal.cpp” se ejecutará automáticamente, en modo paralelo a la aplicación en Matlab, al abrir la interfaz gráfica datos.m, por medio del comando:

! /home/humanoide/Dani/buena/principal&

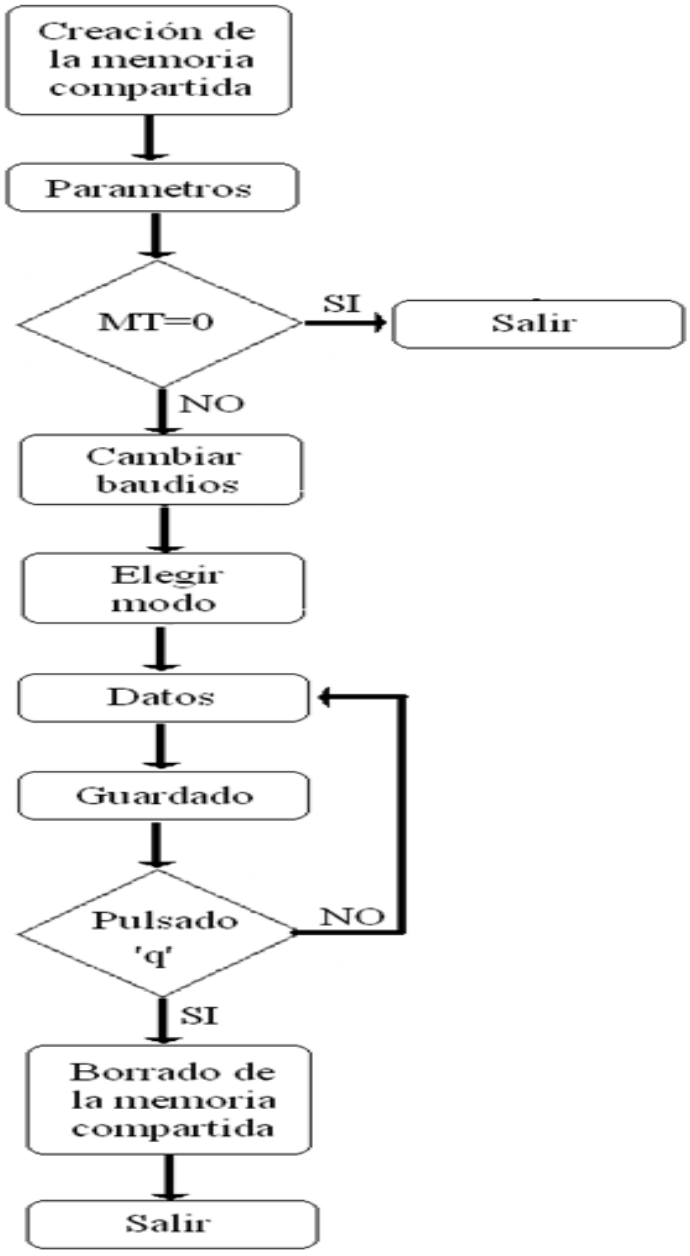

Figura 31: Diagrama de flujo del programa principal.cpp

El diagrama de flujo se explica a continuación:

- Se crea la zona de memoria compartida donde se guardarán los datos, tanto del sensor como los introducidos por el usuario.

```
if((shmid = shmget(CLAVE_SHM,sizeof(info),IPC_CREAT|0666)) == -1)
    {
    perror("shmget");
    exit(EXIT_FAILURE); // Error al crear la memoria compartida
    }
```

```
sensor = (info *)shmat(shmid,0,0); // Obtención del puntero a la estructura de datos compartida
```

- A continuación, se llama a la función "parámetros" que devolverá el valor de MT que indica si el sensor está conectado, y si así fuera, la velocidad de transmisión en baudios y su ID.

```
int parametros() //Saca los MT conectados, la velocidad en baudios y el ID
{   xsens::cmtScanPorts(portInfo);        //Se intenta conectar al xsens
    portCount = portInfo.length(); //Se almacena los MT conectados
```

- Si el valor de MT es distinto de 0, quiere decir, que hay por lo menos 1 sensor conectado, por lo tanto, se puede continuar. Si no fuera así el programa terminaría.

```
if (mtCount == 0) // Asi sabes si hay algun dispositivo conectado
{   cmt3.closePort(); // Cerrar el puerto
    shmctl(shmid,IPC_RMID,0);   // Borrado de la zona de memoria compartida
    return 0;}
```

- El usuario elige la velocidad de conexión en la función "cambio_baudios" a través del la interfaz de configuración, si el usuario no eligió ninguna velocidad, el sensor se conectará a 115200 baudios. Para conectarse a la nueva velocidad el sensor se vuelve a iniciar por lo tanto se vuelve a llamar a la función "parámetros" para asegurarse que se ha conectado a la velocidad querida.

```
void cambio_baudios()
{   //El switch es para saber que baudios ha mandado el usuario
    switch (sensor->baud_man) {
            case 1: baudios=CMT_BAUD_RATE_19K2;  break;
            case 2: baudios=CMT_BAUD_RATE_57K6;  break;
            case 3: baudios=CMT_BAUD_RATE_230K4; break;
            case 4: baudios=CMT_BAUD_RATE_921K6; break;
            default:baudios=CMT_BAUD_RATE_115K2; break; }

    res = cmt3.setBaudrate (baudios, reconnect);// cambiar velocidad baudios
    cmt3.closePort(); // Cerrar el puerto para volverlo a abrir con la nueva velocidad
    parametros();   // Vuelve a abrir el puerto con la nueva velocidad

    switch (portInfo[0].m_baudrate) {
            case B19200 :{ baud_real= 19200;}   break;
            case B38400 :{ baud_real= 38400;}   break;
```

```
case B57600 :{ baud_real= 57600;}   break;
case B115200:{ baud_real=115200;}break;
case B230400:{ baud_real= 230400;}  break;
case B921600:{ baud_real= 921600;}  break;
default: printw("0x%lx", portInfo[0].m_baudrate);
```

- A continuación se elige el modo de salida de datos, para eso, se llama a la función "elegir_modo", en este caso, se ha configurado el sensor para sacar los datos de temperatura, calibración y orientación en los ángulos de Euler.

```
void elegir_modo()
{   modo = CMT_OUTPUTMODE_TEMP | CMT_OUTPUTMODE_CALIB |
    CMT_OUTPUTMODE_ORIENT;
    opcion = CMT_OUTPUTSETTINGS_ORIENTMODE_EULER;
```

- En la función "datos", el sensor envía los paquetes de datos que hemos pedido anteriormente

```
void datos()
{ caldata = packet->getCalData(0); // Array donde se guardan los datos de
                                   aceleracion, giro y campo magnético
    euler_data = packet->getOriEuler(0); // Array donde se guardan los ángulos de
    Euler
```

- Los datos se guardan en la memoria compartida con la función "guardado".

```
void guardado()
{   sensor->acc_x = caldata.m_acc.m_data[0];
```

- Por último si se quiere salir del programa se deberá pulsar la 'q' que borrara la memoria compartida, cerrará los puertos de comunicación y terminará el programa, en caso contrario el programa volverá a la función "datos", luego a "guardado" y se repetirá el bucle constantemente. Esto es verdad, si se ejecuta el programa desde la consola, cuando el programa principal.cpp lo ejecuta la interfaz gráfica (que es la forma habitual) se cerrará cuando la interfaz se cierre, que "matará" el proceso y ejecutara el programa borrado.mexglx para borrar la zona de memoria compartida.

```
while(getch() != 'q')
{   datos();   //Función para obtener los datos del sensor
    guardado(); //Función para guardar los datos del sensor en la memoria
                compartida
}
```

```
cmt3.closePort();
shmctl(shmid,IPC_RMID,0); //Borrado de la zona de memoria compartida
```

4.3 PROGRAMACIÓN UNION.MEXGLX

El programa unión, es el encargado de la comunicación entre el programa C++ y la interfaz gráfica, este programa es imprescindible ya que la interfaz gráfica no puede comunicarse directamente con el programa en C++, ya que Matlab no lo permite.

Para compilar el programa y convertirlo en un archivo MEX se ha utiliza el siguiente comando desde el modo consola en la carpeta donde esté nuestro programa que queremos convertir.

/home/humanoide/matlab/bin/mex union.cpp

La primera parte corresponde a la ubicación de la librería mex.h, en nuestro PC, y la segunda parte es el programa en C++ que queremos compilar.

Este programa tiene una zona de memoria compartida que servirá de nexo de unión entre los dos programas. El programa se estructura de la siguiente manera:

1. Se hace referencia a las librerías que se van a usar, dos librerías fundamentales en este programa son:

```
#include"mex.h"// Librería para generar el fichero.mexglx
#include"memoria.h"// Librería donde están los datos de la memoria
                    compartida
```

2. A continuación se declarará la función principal

```
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
```

3. Luego se declaran las variables que se utilizaran, se va a poner un ejemplo de las variables necesarias para mandar el dato de la aceleración en el eje X y para recibir los baudios a los cuales se debe conectar el sensor

```
double *a;  //Esta variable se utilizará para guardar la dirección de memoria
              correspondiente a plhs[0]
double acc_x; // Aquí se guardará la aceleración en el eje X
int baudios; // Aquí se guardará los baudios
```

4. También se declararán las variables correspondientes a la memoria compartida

```
int shmid;
info *sensor;
```

5. Una vez declaradas las funciones se asignan los valores

```
sensor = (info *)shmat(shmid,0,0);// Obtención del puntero a la estructura de
                                  datos compartida
acc_x=sensor->acc_x;
```

6. Se asigna a cada plhs[] el valor correspondiente que debe mostrar

```
plhs[0]= mxCreateDoubleMatrix(1,1, mxREAL); //Se crea la posición
a = mxGetPr(plhs[0]);   //Se guarda la dirección de la posición
*(a) = acc_x;          //Se guarda el valor en la dirección dada
```

7. Por último se guarda el valor de los baudios requeridos y terminará el programa

```
baudios=(int)mxGetScalar(prhs[0]);
sensor->baud_man=baudios;
```

4.4 INTERFAZ GRÁFICA

Conocida también como GUI (del inglés '*graphical user interface*') es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

En este proyecto, se han realizado una serie de interfaces que son:

- **Sensor:** Ventana de inicia para poder elegir configurar o mostrar datos
- **Configuración:** Configura la velocidad de conexión
- **Datos:** Muestra los datos pedidos

La interfaz datos contiene además de su programación el programa **cubo** que es el encargado de realizar un dibujo aproximado del sensor e imitar el movimiento de éste en tiempo real.

A continuación, en la Figura 32 se muestra un diagrama del desarrollo de la interfaz.

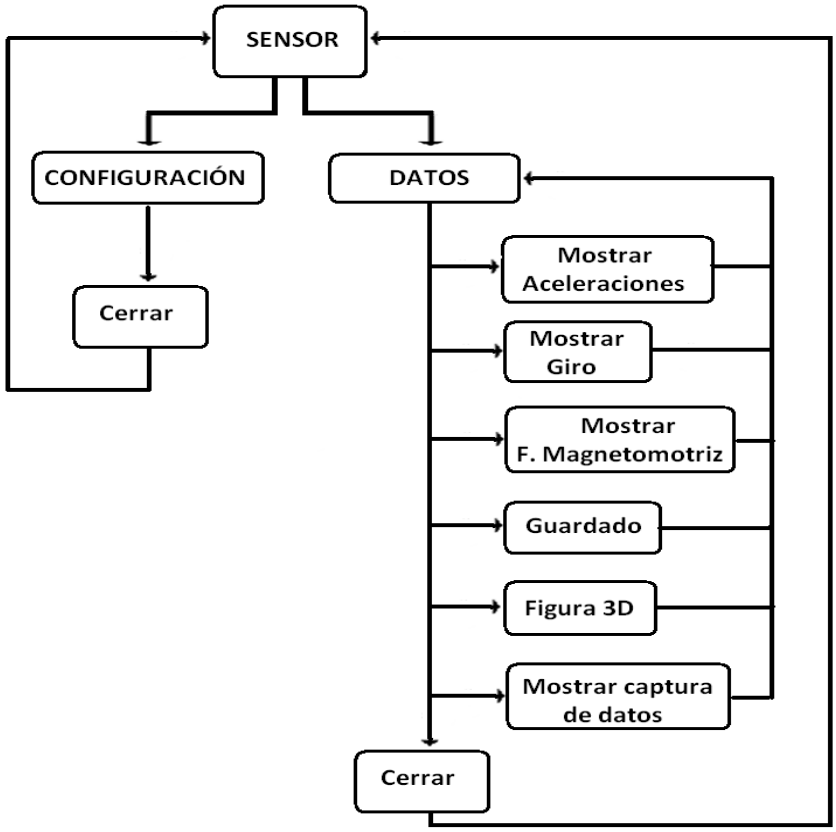

Figura 32: Diagrama de la interfaz gráfica

4.4.1 Sensor

Ésta es la interfaz de arranque, en ella, el usuario tendrá dos opciones (como se puede observar en la Figura 33), configurar y datos [7].

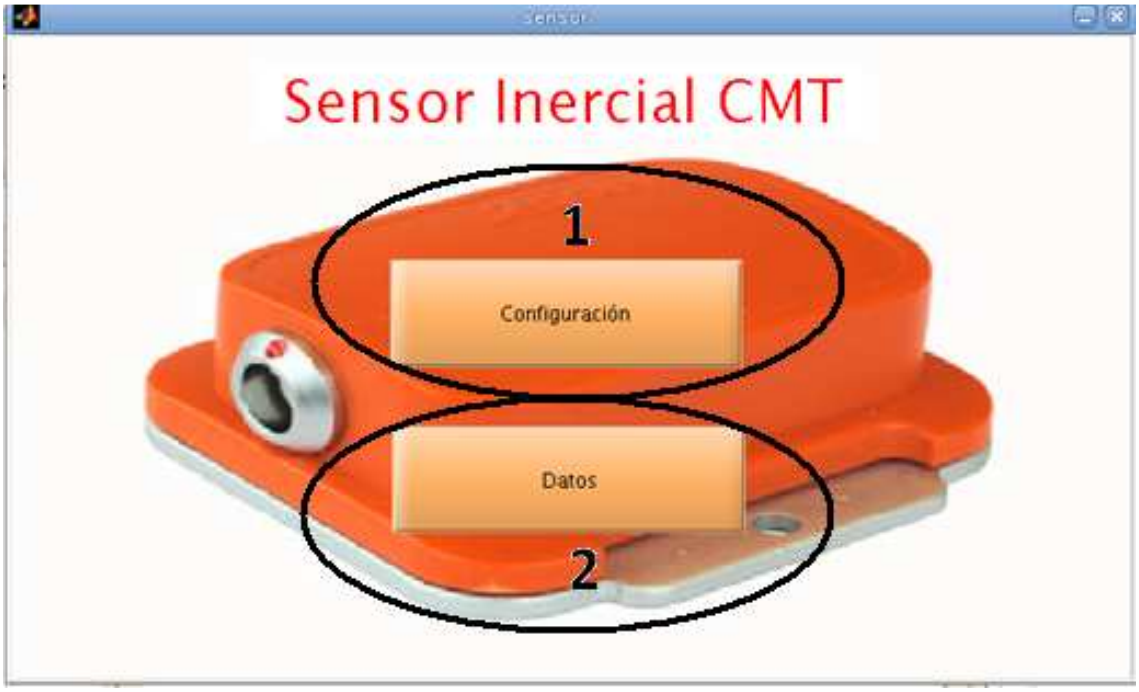

Figura 33: Interfaz gráfica sensor.m

1. **Configuración:** Si se pulsa el botón "Configuración" se llamará a su interfaz gráfica donde se podrá configurar la velocidad de transmisión.

```
% --- Se ejecuta al presionar el botón configuración.
function configuracion_Callback(hObject, eventdata, handles)
configuracion;
```

2. **Datos:** Al pulsar el botón "Datos" se pasará a la interfaz de medición donde se obtendrán los datos pedidos por el usuario.

```
% --- Se ejecuta al presionar el botón datos.
function datos_Callback(hObject, eventdata, handles)
datos;
```

4.4.2 Configuración

La interfaz es la mostrada en la Figura 34, en la cual el usuario podrá elegir los baudios a los que debe conectarse el sensor y dependiendo de éstos el tiempo de muestreo variará según la Tabla 13. Esto es así, para poder darle tiempo al sensor a mandar el mensaje y evitar que el sensor se sature.

Figura 34: Interfaz gráfica configuración.m

Los datos para el modo sin calibrar son de 20 bytes según la Figura 15 y los datos de orientación con los ángulos de Euler son de 12 bytes (ver Figura 12).

Por lo tanto, resolviendo la ecuación 2.1, el tiempo de transmisión se muestra en siguiente tabla.

Tabla 13: Tiempo de transmisión

Baudios	Velocidad de transmisión (ms)
19200	16.7
57600	5.6
115200	2.8
230400	1.4
921600	0.3

Cuando se pulsa cualquiera de los botones se ejecutará la función del panel

```
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
global baudios frecuencia

if hObject == handles.uno
    frecuencia=0.017;
    baudios=1;
elseif hObject == handles.dos
    frecuencia=0.006;
    baudios=2;
elseif hObject == handles.tres
    frecuencia=0.003;
    baudios=3;
elseif hObject == handles.cuatro
    frecuencia=0.002;
    baudios=4;
else
    frecuencia=0.001;
    baudios=5;
end
```

4.4.2 Datos

Esta es la interfaz más importarte, ya que es, la que al fin al cabo, muestra los datos al usuario. Como se puede observar en la Figura 35 en ella se pueden realizar diversas acciones como ver los datos de aceleración, de velocidad angular (giro) o campo magnético en los tres ejes (x, y, z), guardar los datos, graficarlos o mostrar una figura imitando el movimiento del sensor.

Al ejecutar la función de apertura de la interfaz, incluimos una sentencia para que se ejecute paralelamente el programa principal.cpp que iniciará la comunicación del sensor. Además se mandan los baudios a los que se quiere conectar el usuario.

```
% --- Se ejecuta antes de que la interfaz datos sea visible
function datos_OpeningFcn(hObject, eventdata, handles, varargin)
fondo=imread('fondo.jpg');
image(fondo) %Establece el fondo determinado anteriormente
axisoff
! /home/humanoide/Dani/buena/principal& %Pone en marcha principal.cpp
union(baudios) %Manda los baudios requeridos
```

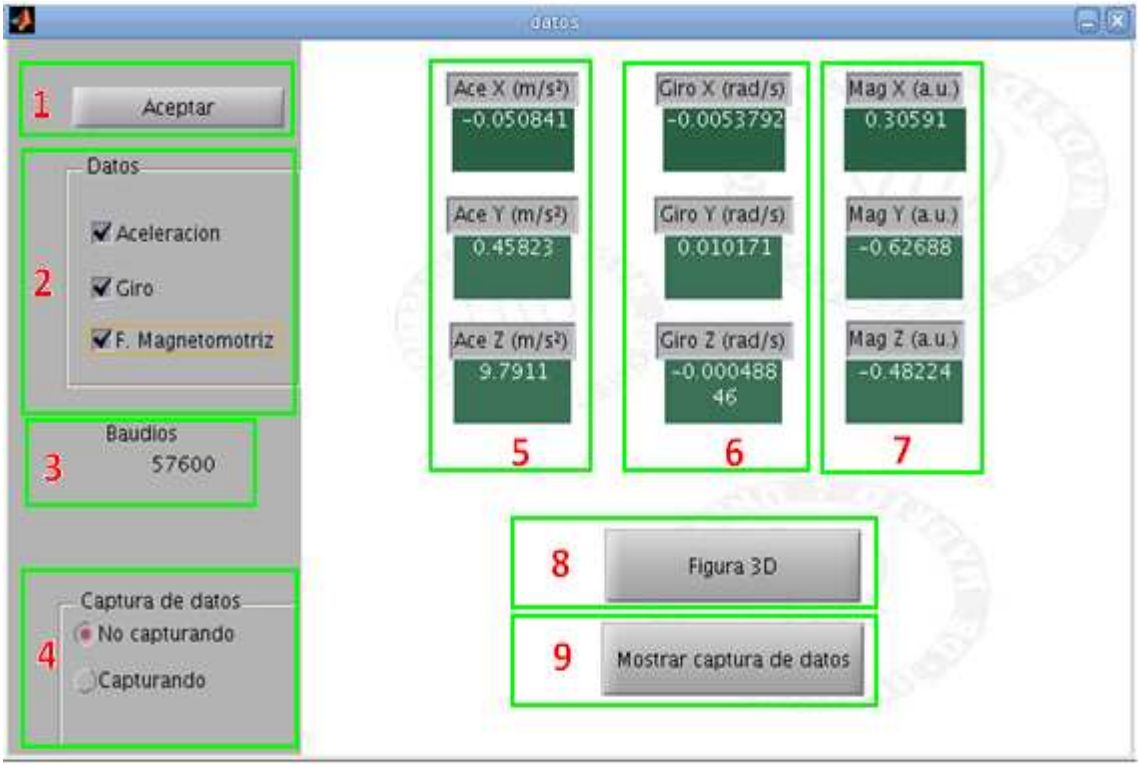

Figura 35: Interfaz gráfica datos.m

Modo de funcionamiento

A continuación describiremos el modo de funcionamiento que tiene la interfaz datos.m:

1. **Aceptar:** Una vez seleccionado los tics que deseamos ver pulsamos el botón para mostrar los datos, habrá que pulsarlo siempre que ya no se estén mostrando otros datos, si fuera así, sólo marcando la casilla correspondiente se mostrarán los datos. Si ya no se desea mostrar alguno de estos datos lo único que se tiene que hacer es desmarcar la casilla correspondiente. La función encargada de realizar esto es:

```
function aceptar_Callback(hObject, eventdata, handles)
% Depende si se ha marcado algún checkbox para realizar el bucle si no es así no hace nada
while salir1==1 || salir2==1 || salir3==1
[acc_x,acc_y,acc_z,gyr_x,gyr_y,gyr_z,mag_x,mag_y,mag_z,baud_real]=union(baudios);
```

2. **Datos:** Se seleccionará que datos queremos mostrar: Aceleraciones, Giro o los Campos magnéticos. Por ejemplo, cuando se pulsa el tic correspondiente a la aceleración se llama a la siguiente función (las otras dos son iguales):

```
function aceleracion_Callback(hObject, eventdata, handles)
global salir1; % Variable que se usa para saber si se ha marcado las aceleraciones
if get(hObject,'Value')% El checkbox ha sido marcado.
    salir1=1;
else
     salir1=0;
end
```

3. **Baudios:** Se mostrarán la velocidad de transmisión, en baudios, a la cual está trabajando el sensor. La instrucción encargada de ello está dentro del bucle de aceptar y es:

```
set(handles.out_baudios,'String', baud_real)
```

4. **Captura de datos:** Si se selecciona la opción Capturando, los datos se guardaran cada uno en su respectiva matriz, una vez que no queramos que se sigan guardando se marcará la opción no capturando. Para poder guardar los datos se tiene que estar mostrando por lo menos un tipo cuando no se muestra ninguno la matriz se reinicia a cero. La función que se ejecuta cuando hay cambios en el panel de selección de captura de datos es:

```
function uipanel2_SelectionChangeFcn(hObject, eventdata, handles)
if hObject == handles.parar
    muestreo=0;  % No se quieren capturar los datos
else
    muestreo=1;  % Se quieren capturar los datos
end
```

Y en el bucle del botón de aceptar se encuentra la instrucción que guardará los datos (se ha puesto el ejemplo de la aceleración en el eje X)

```
if muestreo==1
    accX(i)=int64(acc_x);
    i=i+1;
end
```

5. **Aceleraciones:** Se muestran los datos de Aceleraciones en cada uno de los ejes, si el tic correspondiente a sido marcado, en caso que no lo hubiera sido, estaría en blanco. La instrucción correspondiente para que se lleve a cabo está dentro del bucle de aceptar y es la siguiente:

```
if salir1==1
    set(handles.acc_x, 'String',num2str(acc_x));
else
    set(handles.acc_x, 'String',' ');
end
```

6. **Giro:** Se muestran los datos de velocidad angular en cada uno de los ejes, si el tic correspondiente a sido marcado, en caso que no lo hubiera sido, estaría en blanco. La instrucción correspondiente para que se lleve a cabo está dentro del bucle de aceptar y es la siguiente:

```
if salir2==1
    set(handles.giro_x, 'String',num2str(gyr_x));
else
    set(handles.giro_x, 'String',' ');
end
```

7. **Campo magnético:** Se muestran los datos del campo magnético en cada uno de los ejes, si el tic correspondiente a sido marcado, en caso que no lo hubiera sido, estaría en blanco. La instrucción correspondiente para que se lleve a cabo está dentro del bucle de aceptar y es la siguiente:

```
if salir3==1
    set(handles.mag_x, 'String',num2str(mag_x));
else
    set(handles.mag_x, 'String',' ');
end
```

8. **Figura 3D:** Al pulsar, se mostrará un dibujo del sensor, que se moverá a tiempo real según éste, como el mostrado en la Figura 36. La función correspondiente es:

```
function cubo_Callback(hObject, eventdata, handles)
cubo;
```

9. **Mostrar captura de datos:** Si se pulsa este botón, se mostrarán los datos guardados en tres gráficas (aceleraciones, giro, Campo magnético), y en cada una éstas se representarán los datos de sus ejes (X, Y, Z). La gráfica de las aceleraciones se representa en la función:

```
% --- Se ejecuta al presionar el botón graficar.
function graficar_Callback(hObject, eventdata, handles)
subplot(3,1,1),plot(tiempo,accX,'-r',tiempo,accY,'-g',tiempo,accZ,'-b')
title('accX en rojo; accY en verde; accZ en azul')
xlabel('Tiempo (s)')
ylabel('Aceleraciones (m/s^2)')
```

En la función de cierre de la interfaz datos.m se ejecuta unas sentencias para que el programa principal.cpp termine y otra para borrar la zona de me memoria compartida llamando al programa borrado.mexglx.

```
% --- Se ejecuta cuando el usuario intenta cerrar la interfaz.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
        opc=questdlg('Â¿Desea salir del programa?', 'SALIR','Si','No','No')
if strcmp(opc, 'No')
return;
end
! killall principal& % Cierra el programa principal.cpp
borrado; % Borra la memoria compartida
delete(hObject);
```

A continuación, se explicará más detalladamente, lo que pasa al pulsar el botón “Figura 3D” y “Mostrar captura de datos”.

Al pulsar el boton “Figura 3D”, se ejecuta el programa cubo.m que corresponde con un dibujo aproximado del sensor como se ve en la Figura 36.

Este dibujo se ha realizado uniendo ocho puntos formando un cubo y para que imite el movimiento del sensor se ha multiplcado cada uno de esos puntos por la matriz de Euler como se muestra a continuación, utilizando el punto R1 [6] [8].

```
% Los datos están en grados, hay que pasarlo a radianes para utilizar la matriz de Euler
a=yaw*pi/180;
b=pitch*pi/180;
c=roll*pi/180;
```

```
% R es la matriz de Euler y E1 E2 E3 son sus transformadas
E1=[cos(b), 0, sin(b); 0, 1, 0; -sin(b), 0, cos(b)];
E2=[1, 0, 0; 0, cos(c), -sin(c); 0, sin(c), cos(c)];
E3=[cos(a), -sin(a), 0; sin(a), cos(a), 0; 0, 0, 1];

R=E3*E1*E2;

% Se multiplica cada punto inicial por la matriz de Euler
V1= [x(1);y(1);z(1)];
R1= R*V1;
```

En la siguiente figura se puede observar la siguiente similutud con la Figura 8. El cuadrado azul (R1, R2, R3 y R4) corresponde a la base metálica del sensor y el lado verde (R1, R5, R8 y R4) corresponde a lado donde esta el conector.

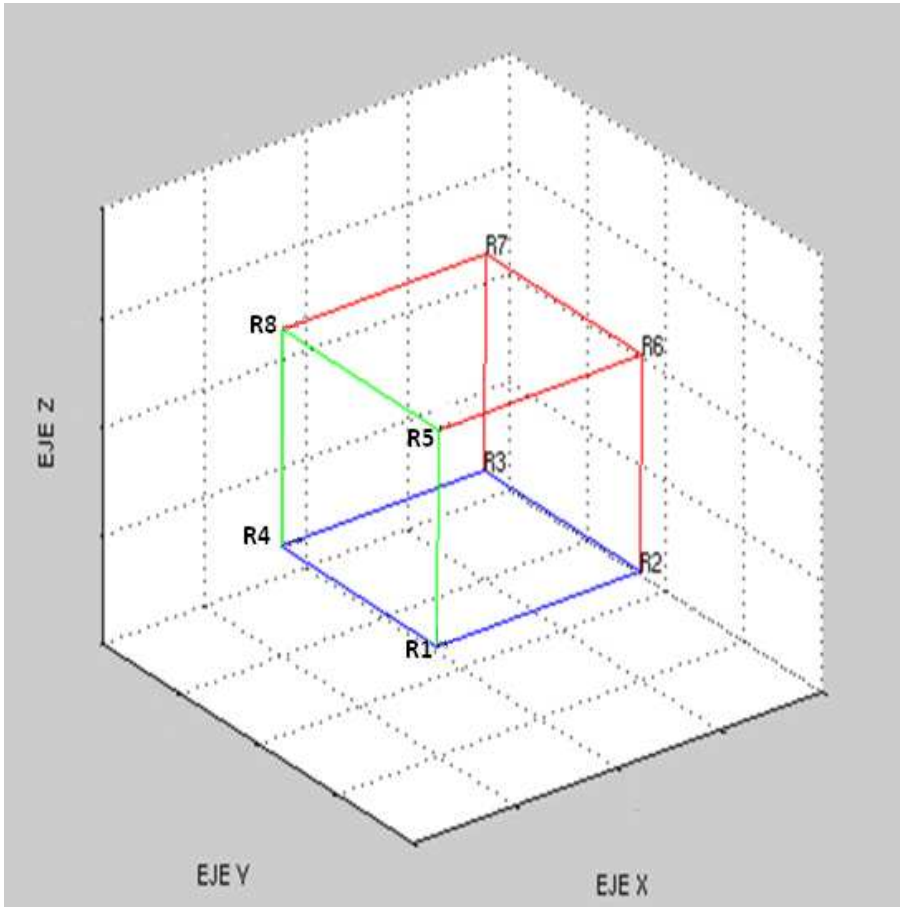

Figura 36: Interfaz gráfica cubo.m

El resultado final se puede observar en la Figura 37, la posición del sensor coincide con su representación gráfica, según se ha explicado anteriormente.

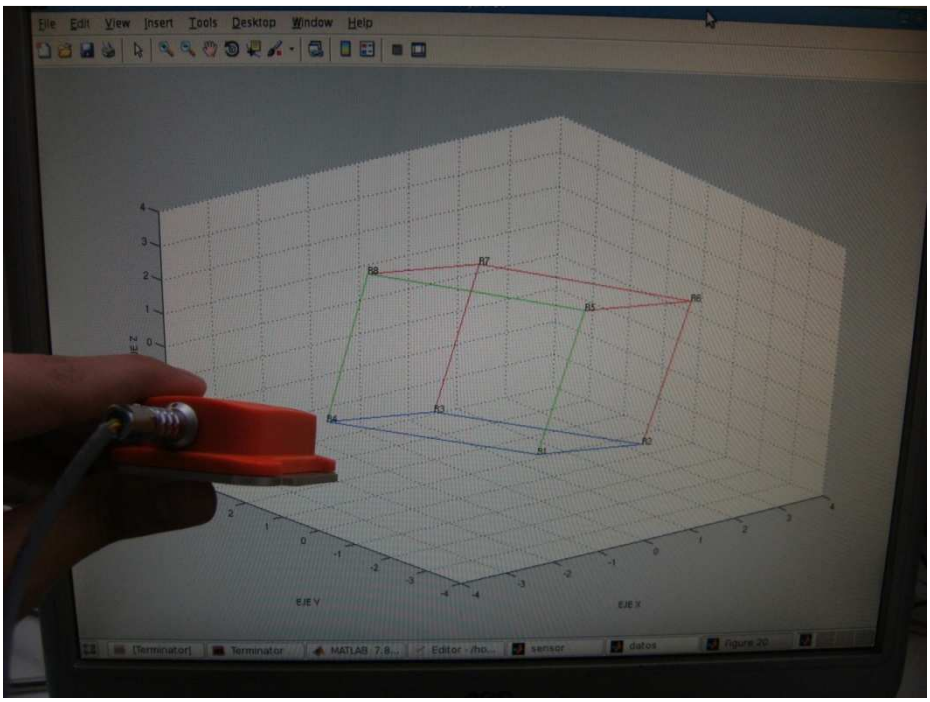

Figura 37: Resultado de la interfaz cubo.m

Pulsando el botón “Mostrar captura de datos” se obtienen las gráficas mostradas en la Figura 38, que se detallarán a continuación:

En la primera gráfica se obtienen las aceleraciones de los tres ejes:

- accX = Aceleración en el eje X, se representa en rojo y con el número 1.
- accY = Aceleración en el eje Y, se representa en verde y con el número 2.
- accZ = Aceleración en el eje Z, se representa en azul y con el número 3.

En la segunda gráfica se obtienen las velocidades angulares (giro) de los tres ejes:

- giroX = Velocidad angular en el eje X, se representa en rojo y con el número 1.
- giroY = Velocidad angular en el eje Y, se representa en verde y con el número 2.
- giroZ = Velocidad angular en el eje Z, se representa en azul y con el número 3.

En la última gráfica se obtienen los campos magnéticos de los tres ejes:

- magX = Campo magnético en el eje X, se representa en rojo y con el número 1.

- magY = Campo magnético en el eje Y, se representa en verde y con el número 2.
- magZ = Campo magnético en el eje Z, se representa en azul y con el número 3.

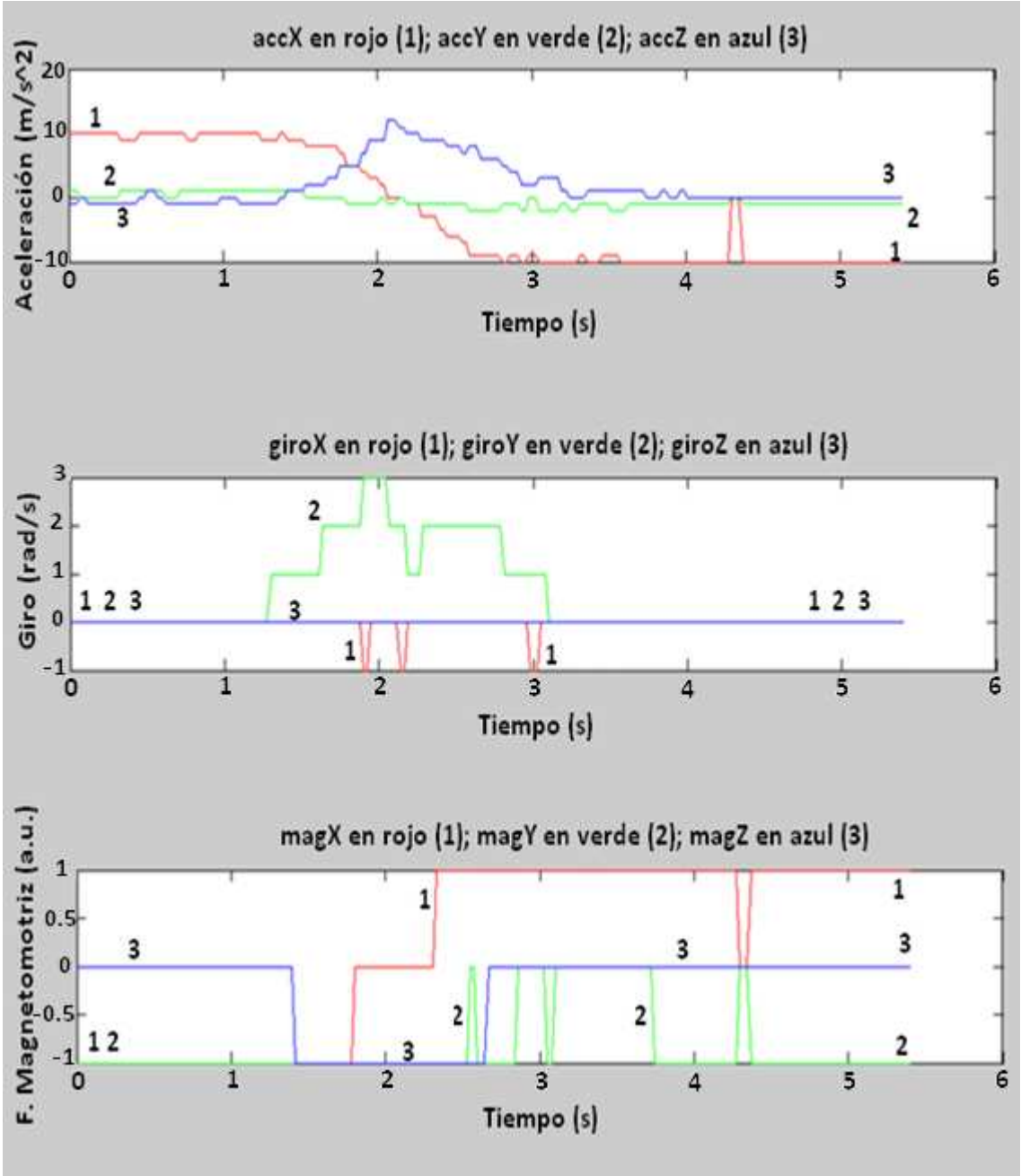

Figura 38: Interfaz gráfica captura

Capítulo 5: RESULTADOS EXPERIMENTALES

A continuación se desarrollaran una serie de pruebas para demostrar que los datos que muestra el sensor son correctos.

5.1 ACELERACIONES

Para comprobar que los datos de las aceleraciones son los correctos se ha utilizado la fuerza de la gravedad, orientando el sensor según sus ejes de coordenadas en la misma dirección que dicha fuerza.

En la Figura 1, al posicionar el sensor con su eje X en vertical se observa que muestra una aceleración de 9.78 m/s^2, la cual se asemeja a la fuerza de la gravedad, por lo tanto se llega a la conclusión de que el sensor actúa correctamente.

Figura 39: Aceleración en el eje X

En la Figura 40 se observa que si se alinea la dirección del eje Y con la fuerza de la gravedad se obtiene 9.87 m/s^2 que es prácticamente la fuerza de la gravedad con un pequeño error debido a que la aceleración de la gravedad no es exactamente en cada lugar.

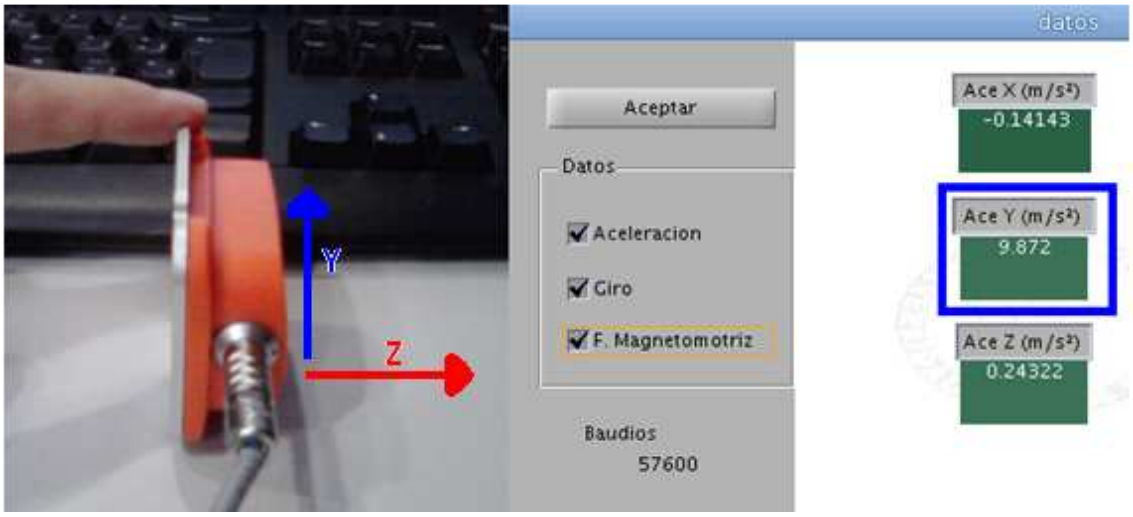

Figura 40: Aceleración en el eje Y

Por último en la Figura 41 se puede ver que la aceleración que está proporcionando el sensor respecto al eje Z es de 9.76 m/s^2, aproximadamente, la aceleración de la gravedad.

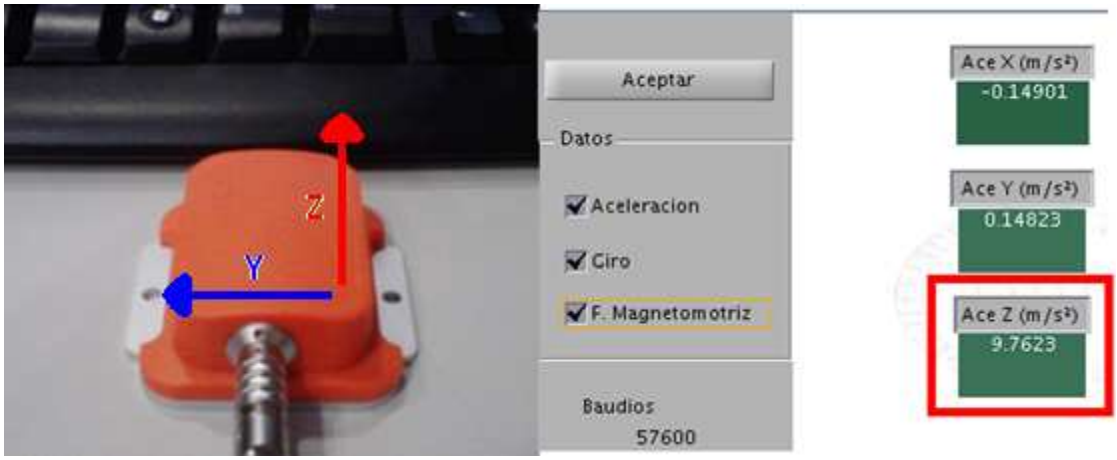

Figura 41: Aceleración en el eje Z

Para terminar, usando la aplicación de captura de datos, se ha posicionado el sensor en las tres posiciones anteriores durante unos seis segundos y se ha obtenido la Figura 42, en la cual, se observa cómo se obtiene la aceleración de la gravedad dependiendo de la posición del sensor.

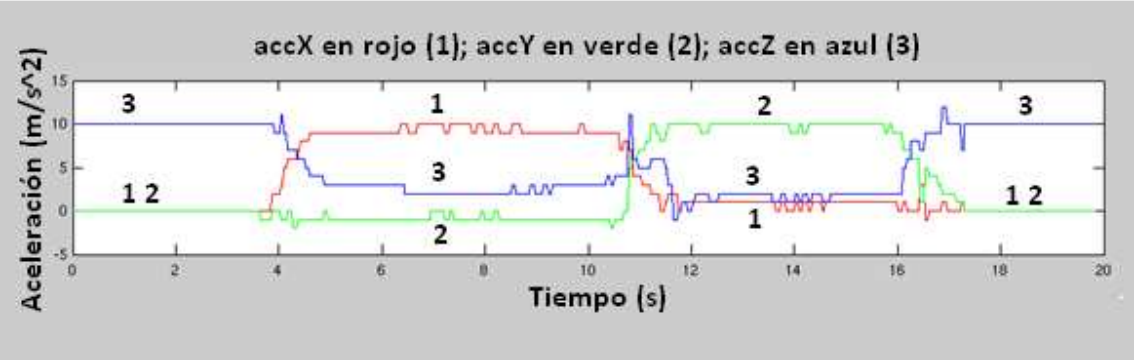

Figura 42: Gráfica de aceleraciones

Todas estas pruebas de la aceleración concluyen que el sensor actúa correctamente en este sentido.

5.2 GIRÓSCOPO

La realización de las pruebas necesarias para saber si la velocidad angular que muestra el sensor es correcta ha sido más complicada. Para poder realizarlas se ha optado por girar el sensor 180^0 (Π rad) a mano.

En primer lugar se ha girado el sensor 180^0 sobre el eje X, dándonos un resultado de aproximadamente 2 rad/s durante un tiempo de unos 1.7s como se puede ver en la Figura 43. Esto da un resultado de 3.4 rad, que se puede aproximar a Π, por lo tanto, es correcta la velocidad obtenida por el sensor.

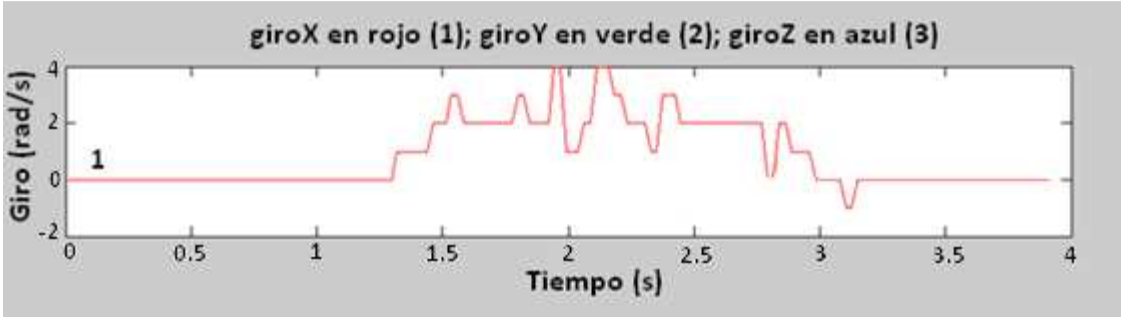

Figura 43: Gráfica de giro en X

A continuación, en la Figura 44, se ha realizado el giro de 180^0 respecto al eje Y, también se obtiene una velocidad aproximada de unos 2 rad/s, durante unos 1.6 s que da como resultado 3.2 rad, con lo que se llega a la conclusión, que es correcta la velocidad tomada por el sensor.

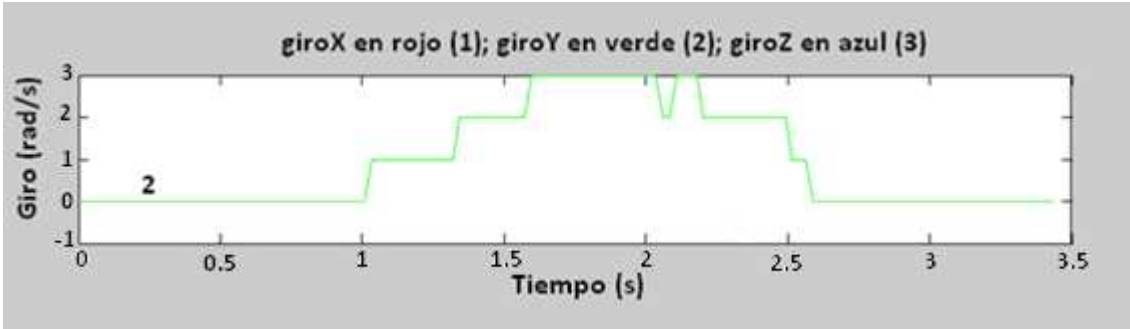

Figura 44: Gráfica de giro en Y

Para terminar esta prueba, se ha girado el sensor en el eje Z. Como se observa en la Figura 34, la medida de la velocidad también ha sido unos 2 rad/s, esto es debido a que se ha intentando mantener la misma velocidad cuando se giraba en cada uno de los ejes. El tiempo que se obtiene es aproximadamente 1.6 s, por lo tanto, da como resultado un giro de 3.2 rad. La medida ha sido correcta.

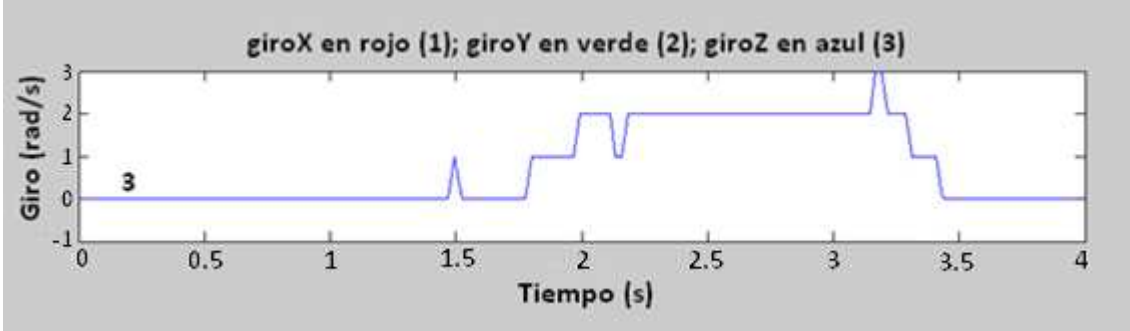

Figura 45: Gráfica de giro en Z

5.3 ÁNGULOS DE EULER

Por último se ha realizado las pruebas para comprobar si los ángulos obtenidos (*'roll', 'pitch', 'yaw'*) son los correctos. Para ello utilizando la aplicación de Figura en 3D, que es la que utiliza dichos ángulos, se han ido tomando distintas imágenes mientras se movía el sensor en cada uno de sus ángulos.

La posición inicial del sensor se observa en la Figura 46.

Figura 46: Posición inicial ángulos de Euler

Y su correspondiente representación se muestra en la Figura 47. Donde el cuadrado azul es la base metálica y el lado verde corresponde al lateral donde está el conector del sensor.

Los ángulos que se girarán serán:

- α es el ángulo entre el eje *x* y la línea de nodos que corresponde a *'yaw'*.
- β es el ángulo entre el eje *z* y el eje *Z* que corresponde a *'rolll'*.
- γ es el ángulo entre la línea de nodos y el eje *X* que corresponde a *'pitch'*.

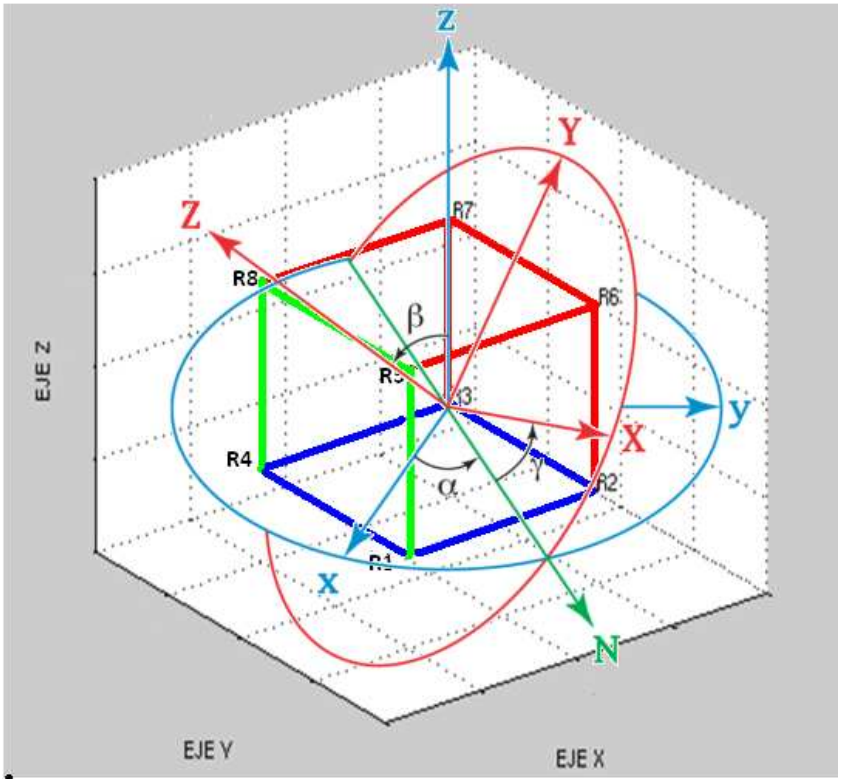

Figura 47: Representación de la posición inicial

Para la primera prueba de la orientación, se ha girado el sensor respecto a su ángulo α= aproximadamente 100^0 (‘*yaw*’), como se muestra en la Figura 48.

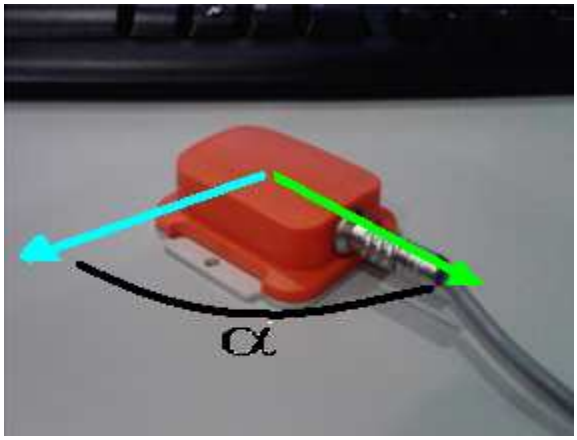

Figura 48: Posición yaw

La figura en 3D se ha movido ese ángulo α como se observa en la Figura 49, pasando de la posición inicial del eje X a N

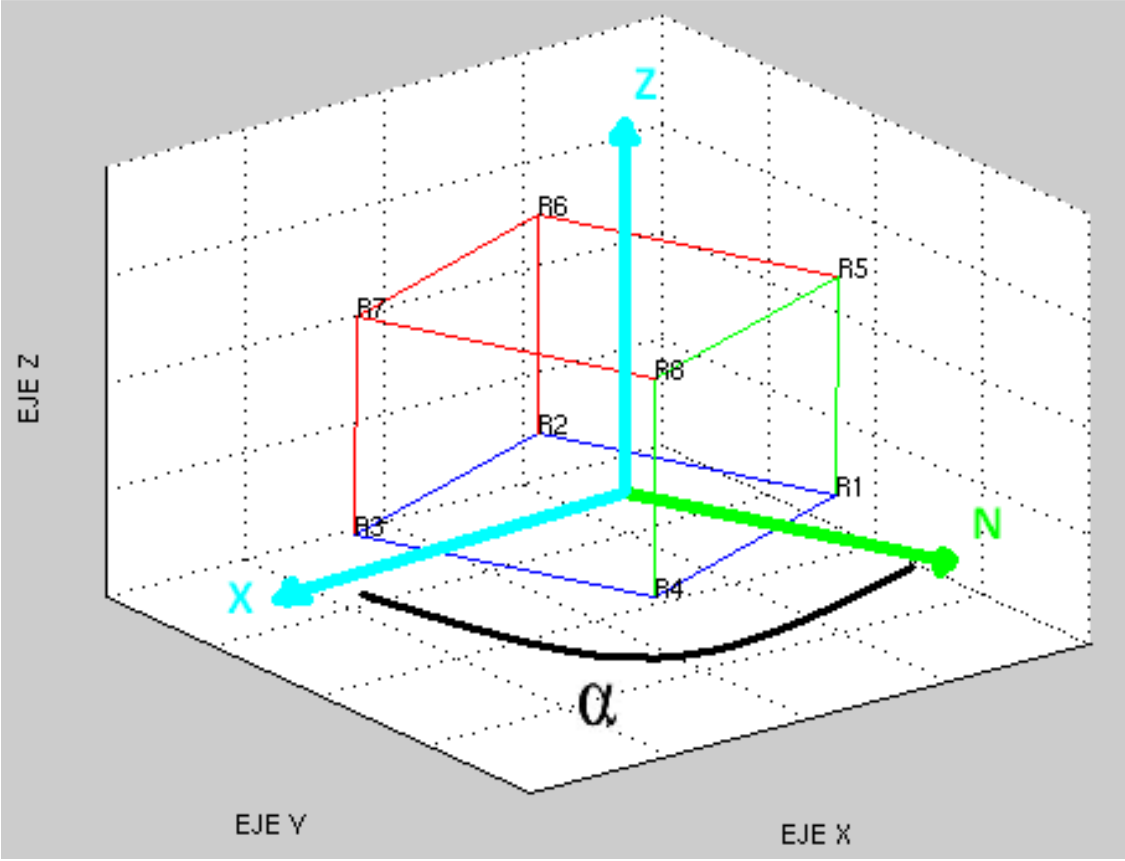

Figura 49: Representación de la posición yaw

A continuación, partiendo de la posición inicial se ha girado el sensor respecto a '*roll*' un ángulo $\beta = 45^0$ como muestra la Figura 50.

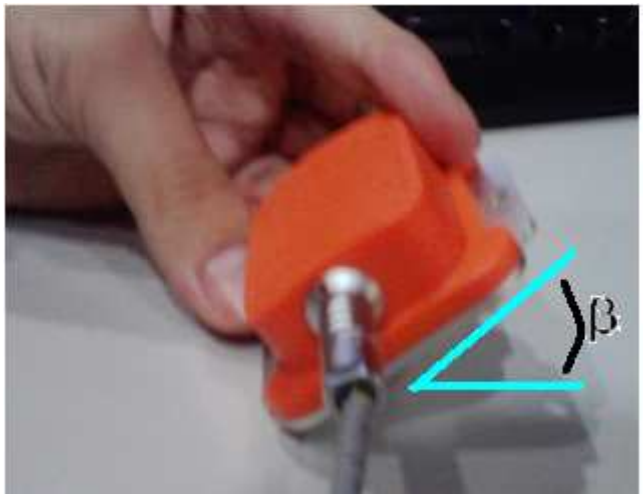

Figura 50: Posición roll

La figura en 3D se ha movido el ángulo β como se puede observar en la Figura 51, cambiando su eje de coordenadas de Z a Z'.

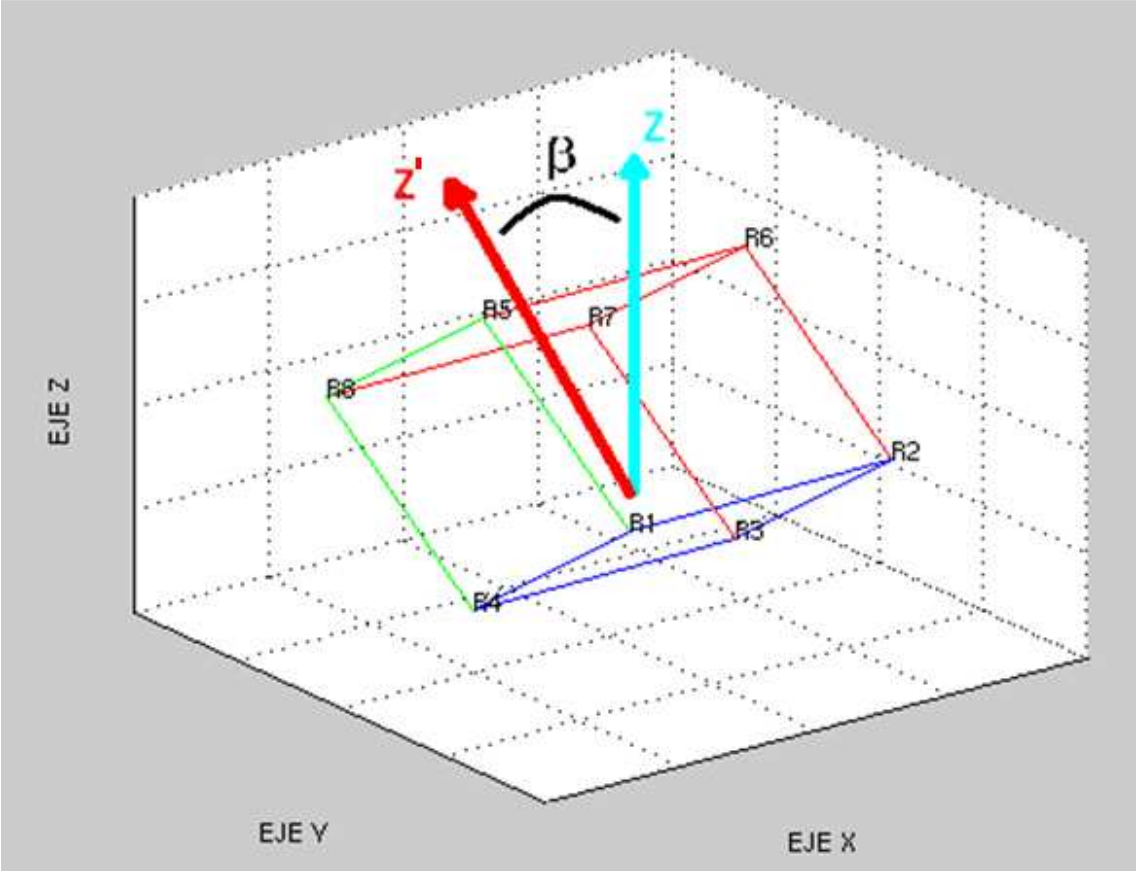

Figura 51: Representación de la posición roll

Por último se giró el sensor en función '*pitch*' un ángulo γ=45^0, como se puede observar en la Figura 52.

Figura 52: Posición pitch

La figura en 3D se ha movido el ángulo γ desde su posición inicial, como se observa en la Figura 53, se desplaza el eje X al X'

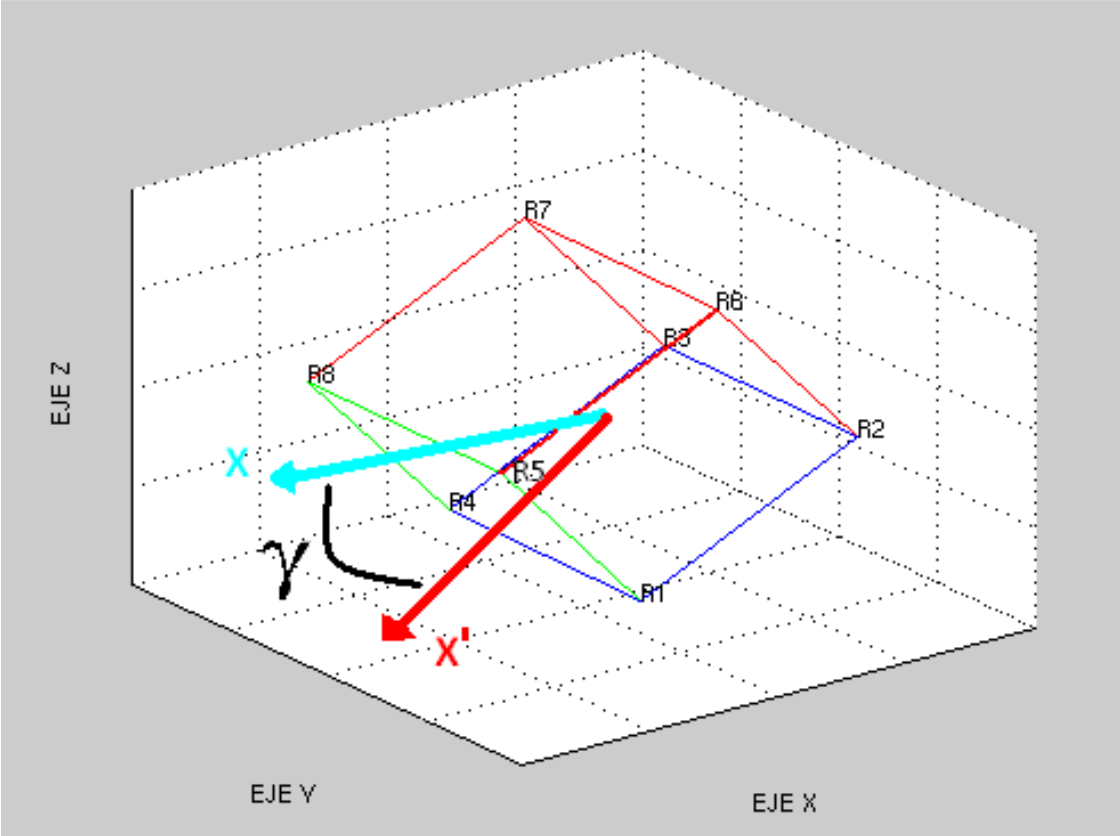

Figura 53: Representación de la posición pitch

Capítulo 6: CONCLUSIONES

6.1 CONCLUSIONES

Todos los objetivos que se plantearon para este proyecto se han cumplido, como se muestra a continuación.

Se ha realizado una comunicación básica con el sensor MTi de la marca Xsens mediante un módulo de comunicación. Este módulo cuenta con un software realizado en el lenguaje de programación C++ para el sistema operativo Linux, llamado "principal.cpp". El programa cuenta con una serie de librerías específicas del sensor que proporciona el fabricante y se ha compilado a través de un makefile para una mayor comodidad.

Para que el usuario pueda comunicarse con el sensor y configurarlo según sus necesidades, se ha desarrollado una serie de interfaces en Matlab basadas en la plataforma Guide, con las cuales, el usuario podrá elegir la velocidad de transmisión, dentro de unas opciones preestablecidas en la interfaz "configuración". Dependiendo de esa velocidad de transmisión el tiempo de muestreo de los datos cambiará. En otra parte de la interfaz, llamada "datos", el usuario tendrá la opción de mostrar los datos de la aceleración, velocidad angular (giro) y campo magnético en cada uno de los ejes del sensor (X, Y, Z). Si se pulsa el botón de "Figura 3D" se generará una interfaz que imitará el movimiento del sensor en tiempo real por medio de los ángulos de Euler. Por último se podrán guardar los datos (si estos se están mostrando) en una serie de matrices para después representarlos en función del tiempo.

Para poder relacionar el módulo de comunicación y la interfaz gráfica se ha creado el programa llamado union.mexglx, el cual se ejecuta desde la interfaz gráfica "datos" en Matlab. Este programa se comunica con principal.cpp a través de una memoria compartida, en la cual, se escribirán los datos mandados por el sensor y la velocidad de transmisión pedida por el usuario.

Por último, se han realizado pruebas en los datos proporcionados por la aceleración orientando la dirección de cada uno de los ejes del sensor con la dirección de la fuerza de la gravedad. También se han realizado un giro controlado en cada uno de sus ejes para comprobar si los datos de velocidad angular son correctos.

6.2 TRABAJOS FUTUROS

- Este proyecto consistía en realizar una comunicación básica con el sensor MTi de la marca Xsens, una vez hecha esta comunicación inicial, se puede ampliar para incluir los demás datos que el sensor puede ofrecer, como por ejemplo, la temperatura que tiene el sensor o la elección del modo calibrado o sin calibrar.

- En este proyecto queda fuera de sus objetivos la ubicación del sensor dentro del humanoide TEO, debido a que está en fase de desarrollo. Una aplicación futura que hay que realizar una vez que esté terminado el robot es la ubicación del sensor dentro del mismo. Las dos posibilidades más adecuadas son en la cabeza o en el torso, en cada una de estas posiciones tendrán sus ventajas y desventajas que se deberán tener en cuenta y ajustar el sensor en función de estas, por ejemplo, si se coloca en la cabeza habrá que ajustar el sensor para detectar si el movimiento que se ha generado es debido al ha movimiento del cuello o debido al desplazamiento del robot entero.

- En nuestra aplicación los datos sólo se pueden guardar si se están mostrando, al menos, uno de los tipos (aceleración, velocidad angular o campo magnético). En una aplicación futura se puede realizar la programación para que guarde los datos offline, sin tener que mostrarse estos por pantalla.

- Una ampliación a este proyecto, debería ser, la posibilidad de elección del modo de visualización de los datos de la orientación en ángulos de Euler, cuaternios o según su matriz de orientación dependiendo de las necesidades del usuario.

- En el modo configuración, la velocidad de transmisión se ha limitado a un rango específico, para poder tener todas las características que ofrece el sensor, este rango se debe ampliar a su capacidad máxima.

Capítulo 7: PRESUPUESTO

1.- Autor: Daniel Morales Tejero

2.- Departamento: Ingeniería de Sistemas y Automática

3.- Descripción del Proyecto

- Titulo: Puesta en marcha del sensor inercial MTi de Xsens
- Duración (meses):4 meses
- Tasa de costes indirectos: Aplicando un 20% : 2000 €

4.- Presupuesto total del Proyecto

14300 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) [1)]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Daniel Morales			1	10000	10000	
					Total	10000

[1)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas). Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [2)]
Ordenador	600	60	3	60	1800
Sensor MTi de Xsens	3000	10	1	60	500
				Total	2300

[2)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	10000
Amortización	0
Subcontratación de tareas	0
Costes de funcionamiento	2300
Costes Indirectos	2000
Total	14300

BIBLIOGRAFÍA

Manuales

- [1] Xsens Technologies. "*MT Low-Level Communication Protocol Documentation*". 2006.
- [2] Xsens Technologies. "*MTi and MTx User Manual and Technical Documentation*". 2006.
- [3] García de Jalón, Javier; Rodríguez, José Ignacio y Vidal, Jesús. "*Aprende Matlab 7.0 como si estuvieras en primero*". Universidad Politécnica de Madrid, 2005.
- [4] Barragán Guerrero, Diego Orlando. "*Manual de Interfaz Gráfica de Usuario en Matlab* ". Universidad Técnica Particular de Loja, 2009

Páginas de internet

- [5]Programación en C++: http://c.conclase.net/ (Mayo / 2010)
- [6]Foro de Matlab: http://www.lawebdelprogramador.com/foros/Matlab/index1.html (Noviembre / 2010)
- [7] Video tutoriales de programación de GUI: http://www.youtube.com/user/diegokillemall (Febrero / 2011)
- [8] Ángulos de Euler: http://es.scribd.com/doc/36263964/18/Rotaciones-matriz-de-Euler (Marzo / 2011)

Proyectos fin de carrera

- [9] García Sánchez, Daniel. "*Sns_Xsens: La integración del sensor inercial MTi dentro de la arquitectura software YARP*". Fecha de lectura Diciembre 2010 en la Universidad Carlos III de Madrid.

- [10] Funes Romero, Gonzalo. "*Diseño e implementación de los sistemas electrónicos del tren inferior del robot humanoide RH-2*". Fecha de lectura Septiembre 2010 en la Universidad Carlos III de Madrid.

- [11] Gallardo Fernández, Guillermo. "*Caracterización y Control de los Motores del Tobillo del Robot Humanoide RH-2*". Fecha de lectura Julio 2010 en la Universidad Carlos III de Madrid.

- [12] Ferrer Mínguez, Gonzalo. "*Integración Kalman de sensores inerciales INS con GPS en un UAV*". Fecha de lectura Abril 2009 en la Universitat Politècnica de Cataluyna.

ANEXOS

A.1 CÓDIGOS DE LOS PROGRAMAS

A continuación se describirán los códigos implementados en la aplicación [5].

A.1.1 Función principal.cpp

```
#include"funcion.h"//Aquí se concentran las librerias que se van a utlizar y las variables
                    globales
int main()
{
//Creación de la zona de memoria compartida
if((shmid = shmget(CLAVE_SHM,sizeof(info),IPC_CREAT|0666)) == -1)
        {
        perror("shmget");
        exit(EXIT_FAILURE); // Error al crear la memoria compartida
        }
  sensor = (info *)shmat(shmid,0,0); // Obtención del puntero a la estructura de datos
                                        compartida

initscr(); //Inicializa todas las estructuras de datos privadas de la biblioteca

parametros(); //Saca los MT conectados, la velocidad en baudios y el ID

if (mtCount == 0) // Asi sabes si hay algun dispositivo conectado
        {       printw("Pulsa la q para salir\n");
                while(getch() != 'q');
                endwin(); // Restaura los valores apropiados de las variables de entorno
                          y sigue trabajando en el modo texto normal.
                cmt3.closePort(); // Cerrar el puerto
                shmctl(shmid,IPC_RMID,0);   // Borrado de la zona de memoria
                                               compartida
                return 0;
        }
sleep(1); //Para que de tiempo a la guide a mandar la nueva velocidad de transimisión.

cambio_baudios();     //Función que configura la velocidad de transimsión

elegir_modo();  //Función que configura el modo de salida de datos del sensor
```

```
while(getch() != 'q')
        {
        datos();   //Función para obtener los datos del sensor
        guardado(); //Función para guardar los datos del senor en la memoria
                    compartida
        }

        endwin();
        cmt3.closePort();
        shmctl(shmid,IPC_RMID,0); //Borrado de la zona de memoria compartida
return 0;
}

int parametros()  // Saca los MT conectados y la velocidad de transmisión
{
        unsignedlong portCount = 0;   //Variable donde se almacena los MT conectados
        xsens::cmtScanPorts(portInfo); //Se intenta conectar al xsens
        portCount = portInfo.length(); //Se almacena los MT conectados

        if (portCount == 0)    //No hay ningún MT conectado
        {       printf("No hay MotionTranckeers conectados\n");
                return 0;
        }

        for(int p = 0; p < (int)portCount; p++)  // Abriendo el puerto
        {
        res = cmt3.openPort(portInfo[p].m_portName, portInfo[p].m_baudrate);
        EXIT_ON_ERROR(res,"cmtOpenPort");  //Mensaje de error si no se ha podido
                                            mandar el mensaje
        }

        for(int j = 0; j <(int)portCount; j++)    // Recibiendo el ID del dispositivo MT
        {
                res = cmt3.getDeviceId((unsignedchar)(j+1), deviceIds[j]);
                EXIT_ON_ERROR(res,"getDeviceId"); //Mensaje de error si no se ha
                                                    podido mandar el mensaje
        }
```

```
        mtCount = cmt3.getMtCount(); // Se guardan los MotionTrackers conectados
        frecuencia = cmt3.getSampleFrequency();   //la frecuencia de conexión

return 0;
}

void cambio_baudios()
{
        bool reconnect; //Necesario para mandar el mensaje para cambiar los baudios
        int baudios;

        //El switch es para saber que baudios ha mandado el usuario
        switch (sensor->baud_man) {
                case 1: baudios=CMT_BAUD_RATE_19K2;  break;
                case 2: baudios=CMT_BAUD_RATE_57K6;  break;
                case 3: baudios=CMT_BAUD_RATE_230K4; break;
                case 4: baudios=CMT_BAUD_RATE_921K6; break;
                default:baudios=CMT_BAUD_RATE_115K2; break;
                        }

        res = cmt3.setBaudrate (baudios, reconnect);// cambiar velocidad baudios
        EXIT_ON_ERROR(res,"baudios");  //Mensaje de error si no se ha podido mandar
                                          el mensaje
        cmt3.closePort(); // Cerrar el puerto para volverlo a abrir con la nueva
                             velocidad
        parametros();   // Vuelve a abrir el puerto con la nueva velocidad

                switch (portInfo[0].m_baudrate) {
                case B9600  :{ baud_real= 9600;}    break;
                case B19200 :{ baud_real= 19200;}   break;
                case B38400 :{ baud_real= 38400;}   break;
                case B57600 :{ baud_real= 57600;}   break;
                case B115200:{ baud_real=115200;}break;
                case B230400:{ baud_real= 230400;}  break;
                case B460800:{ baud_real= 460800;}  break;
                case B921600:{ baud_real= 921600;}  break;
                default: printw("0x%lx", portInfo[0].m_baudrate);
                }
}
```

```
void elegir_modo()
{
	CmtOutputMode modo;   // variable para guardar el modo
	CmtOutputSettings opcion; // variable para guardar la opcion (euler o
	                          cuaternios)

	modo = CMT_OUTPUTMODE_TEMP | CMT_OUTPUTMODE_CALIB |
	CMT_OUTPUTMODE_ORIENT;
	opcion = CMT_OUTPUTSETTINGS_ORIENTMODE_EULER;

	res = cmt3.gotoConfig();   //mensaje para ir al modo configuración
	EXIT_ON_ERROR(res,"gotoConfig"); //Mensaje de error si no se ha podido
	                                 mandar el mensaje

	unsignedshort sampleFreq;
	sampleFreq = cmt3.getSampleFrequency();  // Se guarda la frecuencia del
	                                         sensor

	CmtDeviceMode deviceMode(modo, opcion, sampleFreq); // Establece el modo
	                                                    de salida del dispositivo
		if ((deviceIds[0] & 0xFFF00000) != 0x00500000) {
			// no hay un MTi-G, quita todo lo realacionado con GPS
			deviceMode.m_outputMode &= 0xFF0F;
		}
	res = cmt3.setDeviceMode(deviceMode, true, deviceIds[0]);// Se manda la
	                                                         configuracion de salida
	EXIT_ON_ERROR(res,"setDeviceMode");//Mensaje de error si no se ha podido
	                                   mandar el mensaje
	res = cmt3.gotoMeasurement();   // Se manda el mensaje para empezar a
	                                recibir los datos
	EXIT_ON_ERROR(res,"gotoMeasurement");//Mensaje de error si no se ha
	                                     podido mandar el mensaje
}

void datos()
{
	double tdata; //Variable para guardar la temperatura

	Packet* packet = new Packet(1, cmt3.isXm()); //Se crea una estructura con el
	                                             formato adecuado para guardar los datos
```

```
        cmt3.waitForDataMessage(packet);  //Se guardan todos los datos en la
                                                estructura packet
        tdata = packet->getTemp(0);  // Dato de la temperatura (si quieres mostrarla)
        caldata = packet->getCalData(0); // Array donde se guardan los datos de
                                                aceleracion, giro y campo magnético
        euler_data = packet->getOriEuler(0); // Array dond se guardan los angulos de
                                                euler
}

void guardado()
{
//Se guardaran todos los datos en la memoria compartida para luego mostrarlos
        sensor->acc_x = caldata.m_acc.m_data[0];
        sensor->acc_y = caldata.m_acc.m_data[1];
        sensor->acc_z = caldata.m_acc.m_data[2];

        sensor->gyr_x = caldata.m_gyr.m_data[0];
        sensor->gyr_y = caldata.m_gyr.m_data[1];
        sensor->gyr_z = caldata.m_gyr.m_data[2];

        sensor->mag_x = caldata.m_mag.m_data[0];
        sensor->mag_y = caldata.m_mag.m_data[1];
        sensor->mag_z = caldata.m_mag.m_data[2];

        sensor->baud_rec= baud_real;

        sensor->roll = euler_data.m_roll;
        sensor->pitch= euler_data.m_pitch;
        sensor->yaw= euler_data.m_yaw;
}
```

A.1.2 Librería funcion.h

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/ioctl.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

```
#include<fcntl.h>
#include<signal.h>
#include<curses.h>

//Librerias del sensor MT
#include"cmtdef.h"
#include"xsens_time.h"
#include"xsens_list.h"
#include"cmtscan.h"
#include"cmt3.h"
#include"example_linux.h"

#include"memoria.h"//Aquí se guardaran los datos de la memoria compartida

//Configuración de los mensajes de error
#define EXIT_ON_ERROR(res,comment) if (res != XRV_OK) { printf("Error %d occurred
in " comment ": %s\n",res,xsensResultText(res)); exit(1); }

usingnamespace xsens; //Para escribir funciones mas concisas

//Declaración de las funciones que se utilizaran
int parametros();
void datos();
void guardado();
void elegir_modo();
void cambio_baudios();

xsens::Cmt3 cmt3;
int mtCount = 0; // Se guardaran cuantos MT hay conectados al equipo
short frecuencia; //Variable para guardar la frecuencia de muestreo
int baud_real; // Se guardaran los baudios a los que esta conectado el sensor

CmtDeviceId deviceIds[256]; //Para guardar la ID del dispositivo

List<CmtPortInfo> portInfo;  // Aqui se guardara el nombre del puerto COM, los
                                baudios
XsensResultValue res = XRV_OK;

CmtCalData caldata;//Estructura del tipo adecuada para guardar los datos
CmtQuat qat_data; //Estructura del tipo adecuada para gaurdar los cuaternios
```

```
CmtEuler euler_data; //Estructura del tipo adecuada para guardar los ángulos de Euler
CmtMatrix matrix_data; //Estructura del tipo adecuada para guardar la matriz de datos

info *sensor; //Puntero a una estructura tipo info
int shmid; //Se guardará el identificador de la memoria compartida

size_t aux;
```

A.1.3 Librería memoria.h

```
/* Fichero memoria.h, contiene información sobre la zona de memoria que se
pretende compartir entre diversos procesos, será preciso incluirlo en la cabecera del
fichero de aquellos procesos que se conecten a la zona de memoria común. Definición
de la clave de acceso a la zona de memoria común */

#define CLAVE_SHM ((key_t) 1001) //Clave de la memoria compartida

/* Estructura de datos que se pretende compartir en la zona de memoria común */
Typedefstruct
{
float acc_x;
float acc_y;
float acc_z;
float gyr_x;
float gyr_y;
float gyr_z;
float mag_x;
float mag_y;
float mag_z;
int baud_man;
int baud_rec;
float roll;
float pitch;
floatyaw;

}info;
```

A.1.4 Fichero union.mexglx

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/shm.h>
#include<signal.h>
#include<curses.h>
```

```
#include"mex.h"// Libreria para generar el fichero.mexglx
#include"memoria.h"// Libreria donde están los datos de la memoria compartida


void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
double *a, *b, *c, *d, *e, *f, *g, *h, *i, *j, *k, *l, *m;
double acc_x, acc_y, acc_z, gyr_x, gyr_y, gyr_z, mag_x, mag_y, mag_z, roll, pitch, yaw;
int shmid, baudios, baud_rec;
info *sensor;

// Creación de la zona de memoria compartida
if((shmid = shmget(CLAVE_SHM, sizeof(info), IPC_CREAT|0666)) == -1)
        {
        perror("shmget");
        exit(EXIT_FAILURE);
        }

sensor = (info *)shmat(shmid,0,0);// Obtención del puntero a la estructura de datos
                                  compartida

//Variables donde se almacenan los datos de la memoria compartida
acc_x=sensor->acc_x;
acc_y=sensor->acc_y;
acc_z=sensor->acc_z;
gyr_x=sensor->gyr_x;
gyr_y=sensor->gyr_y;
gyr_z=sensor->gyr_z;
mag_x=sensor->mag_x;
mag_y=sensor->mag_y;
mag_z=sensor->mag_z;

baud_rec=sensor->baud_rec;

roll= sensor->roll;
pitch= sensor->pitch;
yaw= sensor->yaw;


//En el array plhs se guardaran los datos de la memoria compartida
plhs[0]= mxCreateDoubleMatrix(1,1, mxREAL); //Se crea la posición
a = mxGetPr(plhs[0]);   //Se guarda la dirección de la posición
*(a) = acc_x;         //Se guarda el valor en la dirección dada

plhs[1] = mxCreateDoubleMatrix(1,1, mxREAL);
b = mxGetPr(plhs[1]);
```

```
*(b) = acc_y;

plhs[2]= mxCreateDoubleMatrix(1,1, mxREAL);
c = mxGetPr(plhs[2]);
*(c) = acc_z;

plhs[3]= mxCreateDoubleMatrix(1,1, mxREAL);
d = mxGetPr(plhs[3]);
*(d) = gyr_x;

plhs[4]= mxCreateDoubleMatrix(1,1, mxREAL);
e = mxGetPr(plhs[4]);
*(e) = gyr_y;

plhs[5]= mxCreateDoubleMatrix(1,1, mxREAL);
f = mxGetPr(plhs[5]);
*(f) = gyr_z;

plhs[6]= mxCreateDoubleMatrix(1,1, mxREAL);
g = mxGetPr(plhs[6]);
*(g) = mag_x;

plhs[7]= mxCreateDoubleMatrix(1,1, mxREAL);
h = mxGetPr(plhs[7]);
*(h) = mag_y;

plhs[8]= mxCreateDoubleMatrix(1,1, mxREAL);
i = mxGetPr(plhs[8]);
*(i) = mag_z;

plhs[9]= mxCreateDoubleMatrix(1,1, mxREAL);
j = mxGetPr(plhs[9]);
*(j) = baud_rec;

plhs[10]= mxCreateDoubleMatrix(1,1, mxREAL);
i = mxGetPr(plhs[10]);
*(i) = roll;


plhs[11]= mxCreateDoubleMatrix(1,1, mxREAL);
i = mxGetPr(plhs[11]);
*(i) = pitch;

plhs[12]= mxCreateDoubleMatrix(1,1, mxREAL);
i = mxGetPr(plhs[12]);
*(i) = yaw;
```

```
baudios=(int)mxGetScalar(prhs[0]); //Devuelve los baudios a los que desea conectar el usuario

sensor->baud_man=baudios;  //Se guardan en la memoria compartida los baudios que quiere tener el usuario
}
```

A.1.5 Fichero borrado.mexglx

```
#include<stdio.h>
#include<sys/shm.h>

#include"mex.h"
#include"memoria.h"

void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
int shmid;
info *sensor;

// Creación de la zona de memoria compartida
if((shmid = shmget(CLAVE_SHM, sizeof(info), IPC_CREAT|0666)) == -1)
        {
          perror("shmget");
         exit(EXIT_FAILURE);
        }

//Obtención del puntero a la estructura de datos compartida
sensor = (info *)shmat(shmid,0,0);

//Borrado de la zona de memoria compartida
shmctl(shmid,IPC_RMID,0);

}
```

A.1.6 Makefile

```
CC=g++
CFLAGS=-I. -Wall
LFLAGS=-lrt
DEPS = cmt1.h
OBJSTATIC = cmt1.o cmt2.o cmt3.o cmtmessage.o cmtpacket.o cmtscan.o xsens_std.o
xsens_time.o

all: static principal

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c -o $@ $<

principal: $(OBJSTATIC) principal.cpp
	$(CC) $(CFLAGS) $(LFLAGS) $(OBJSTATIC) principal.cpp -lncurses -o principal

static: $(OBJSTATIC)
	ar rcs libcmt.a $^

clean:
	rm -f matlab
	rm -f *.o
	rm -f libcmt.a
	rm -f libcmt.so
```

A.2 CÓDIGOS DE LAS INTERFACES

A.2.1 Sensor.m

```
function varargout = sensor(varargin)

% --- Inicialización del código – NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @sensor_OpeningFcn, ...
'gui_OutputFcn',  @sensor_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% --- Final de la inicialización del código – NO EDITAR


% --- Se ejecuta antes de que la interfaz sensor sea visible
function sensor_OpeningFcn(hObject, eventdata, handles, varargin)

fondo=imread('foto_sensor.jpg');
image(fondo)
axisoff

% --- Escoge el comando por defecto de la línea de salida de sensor
handles.output = hObject;

% --- Se actualize la estructura handles
guidata(hObject, handles);

% --- El resultado de esta función se devuelve a la línea de comandos.
function varargout = sensor_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Se ejecuta al presionar el botón configuración.
function configuracion_Callback(hObject, eventdata, handles)
configuracion;
```

```
% --- Se ejecuta al presionar el botón datos.
function datos_Callback(hObject, eventdata, handles)
datos;
```

A.2.2 Configuracion.m

```
function varargout = configuracion(varargin)

% --- Inicialización del código – NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @configuracion_OpeningFcn, ...
'gui_OutputFcn',  @configuracion_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
         [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
         gui_mainfcn(gui_State, varargin{:});
end
% --- Final de la inicialización del código – NO EDITAR


% --- Se ejecuta antes de que la interfaz configuracion sea visible
function configuracion_OpeningFcn(hObject, eventdata, handles, varargin)

fondo=imread('fondo.jpg');
image(fondo)
axisoff

% --- Escoge el comando por defecto de la línea de salida de configuracion
handles.output = hObject;

% --- Se actualice la estructura handles
guidata(hObject, handles);


% --- El resultado de esta función se devuelve a la línea de comandos.
```

```
function varargout = configuracion_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --- Se ejecuta cuando se cambia la opción seleccionada en el  uipanel1.
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
global baudios frecuencia

if hObject == handles.uno
      frecuencia=0.017;
      baudios=1;

elseif hObject == handles.dos
      frecuencia=0.006;
      baudios=2;

elseif hObject == handles.tres
      frecuencia=0.003;
      baudios=3;

elseif hObject == handles.cuatro
      frecuencia=0.002;
      baudios=4;

else
      frecuencia=0.001;
      baudios=5;
end
```

A.2.3 Datos.m

```
function varargout = datos(varargin)

% --- Inicialización del código – NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @datos_OpeningFcn, ...
'gui_OutputFcn',  @datos_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});


end
% --- Final de la inicialización del código – NO EDITAR


% --- Se ejecuta antes de que la interfaz datos sea visible
function datos_OpeningFcn(hObject, eventdata, handles, varargin)
global baudios

fondo=imread('fondo.jpg');
image(fondo)
axisoff
! /home/humanoide/Dani/buena/principal&%Pone en marcha principal.cpp
union(baudios)

%--- Escoge el comando por defecto de la línea de salida de datos
handles.output = hObject;

%---Se actualice la estructura handles
guidata(hObject, handles);

% --- El resultado de esta función se devuelve a la línea de comandos.
function varargout = datos_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Se ejecuta al presionar el botón aceleracion.
function aceleracion_Callback(hObject, eventdata, handles)
global salir1; % Variable que se usa para saber si se ha marcado las aceleraciones

if get(hObject,'Value')% El checkbox ha sido marcado.
  salir1=1;
else
   salir1=0;
end

% --- Executes on button press in giro.
function giro_Callback(hObject, eventdata, handles)
global salir2; % Variable que se usa para saber si se ha marcado los giros
```

```
if get(hObject,'Value')
   salir2=1;
else
   salir2=0;
end

% --- Executes on button press in mag.
function mag_Callback(hObject, eventdata, handles)
global salir3;% Variable que se usa para saber si se ha marcado el Campo magnético

if get(hObject,'Value')
   salir3=1;
else
   salir3=0;
end

%función edit1_Callback(hObject, eventdata, handles)
% --- Se ejectua durante la creación de objetos, después de establecer todas las
propiedades.
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit2_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit2_CreateFcn(hObject, eventdata, handles)


if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit3_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
```

```
%función edit4_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit5_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit6_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit6_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit7_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit8_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit8_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

%función edit9_Callback(hObject, eventdata, handles)
% --- Se ejecuta durante la creación de objetos, después de establecer todas las
propiedades.
function edit9_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

% --- Se ejecuta cuando el usuario intent cerrar la interfaz.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
opc=questdlg('Â¿Desea salir del programa?', 'SALIR','Si','No','No')
if strcmp(opc, 'No')
return;
end
! killall principal&% Cierra el programa principal.cpp
borrado;
delete(hObject);

% --- Se ejecuta al presionar el botón aceptar.
function aceptar_Callback(hObject, eventdata, handles)
global acc_x acc_y acc_z gyr_x gyr_y gyr_z mag_x mag_y mag_z
global salir1 salir2 salir3 baudios baud_real muestreo tiempo frecuencia
global accX accY accZ giroX giroY giroZ magX magY magZ
i=1;
aux=0;

% Depende si se ha marcado algún checkbox para realizar el bucle
while salir1==1 || salir2==1 || salir3==1

pause(frecuencia) % Define el tiempo de muestreo

[acc_x,acc_y,acc_z,gyr_x,gyr_y,gyr_z,mag_x,mag_y,mag_z,baud_real]=union(baudios);

set(handles.out_baudios,'String', baud_real)

if salir1==1
      set(handles.acc_x, 'String',num2str(acc_x));
      set(handles.acc_y, 'String',num2str(acc_y));
      set(handles.acc_z, 'String',num2str(acc_z));
```

```
else
      set(handles.acc_x, 'String','  ');
      set(handles.acc_y, 'String','  ');
      set(handles.acc_z, 'String','  ');
end

if salir2==1
      set(handles.giro_x, 'String',num2str(gyr_x));
      set(handles.giro_y, 'String',num2str(gyr_y));
      set(handles.giro_z, 'String',num2str(gyr_z));

else
      set(handles.giro_x, 'String','  ');
      set(handles.giro_y, 'String','  ');
      set(handles.giro_z, 'String','  ');
end


if salir3==1
      set(handles.mag_x, 'String',num2str(mag_x));
      set(handles.mag_y, 'String',num2str(mag_y));
      set(handles.mag_z, 'String',num2str(mag_z));
else
      set(handles.mag_x, 'String','  ');
      set(handles.mag_y, 'String','  ');
      set(handles.mag_z, 'String','  ');
end

if muestreo==1    % Se quieren capturar los datos
  if aux==0     % Para poder iniciar el tiempo en 0 cuando terminemos
      tic% Empieza a contar el tiempo
      aux=aux+1;
      accX=0;
      accY=0;
      accZ=0;
      giroX=0;
      giroY=0;
      giroZ=0;
      magX=0;
      magY=0;
      magZ=0;
      tiempo=0;
  end

% Guarda los valores en formato de 64 bits para poder representarlas
```

```
  accX(i)=int64(acc_x);
  accY(i)=int64(acc_y);
  accZ(i)=int64(acc_z);
  giroX(i)=int64(gyr_x);
  giroY(i)=int64(gyr_y);
  giroZ(i)=int64(gyr_z);
  magX(i)=int64(mag_x);
  magY(i)=int64(mag_y);
  magZ(i)=int64(mag_z);
  tiempo(i)=toc;

  i=i+1;

end

end %Final del while


% --- Se ejecuta al presionar el botón cubo.
function cubo_Callback(hObject, eventdata, handles)
cubo;

% --- Se ejecuta cuando se cambia la opción seleccionada en el uipanel2.
function uipanel2_SelectionChangeFcn(hObject, eventdata, handles)
global muestreo% Variable para saber si se quiere guardar los datos

if hObject == handles.parar
        muestreo=0;
else
        muestreo=1;
end


% --- Se ejecuta al presionar el botón graficar.
function graficar_Callback(hObject, eventdata, handles)
global accX accY accZ giroX giroY giroZ magX magY magZ tiempo

figure(10)

subplot(3,1,1),plot(tiempo,accX,'-r',tiempo,accY,'-g',tiempo,accZ,'-b')
title('accX en rojo; accY en verde; accZ en azul')
xlabel('Tiempo (s)')
ylabel('Aceleraciones (m/s^2)')

subplot(3,1,2),plot(tiempo,giroX,'-r',tiempo,giroY,'-g',tiempo,giroZ,'-b')
title('giroX en rojo; giroY en verde; giroZ en azul')
```

```
xlabel('Tiempo (s)')
ylabel('Giro (rad/s)')

subplot(3,1,3),plot(tiempo,magX,'-r',tiempo,magY,'-g',tiempo,magZ,'-b')
title('magX en rojo; magY en verde; magZ en azul')
xlabel('Tiempo (s)')
ylabel('F. Magnetomotriz (a.u)')

%Estos comandos son para maximizar la ventana
drawnow% Neceario para envitar errores de Java
jFig = get(handle(10), 'JavaFrame');
jFig.setMaximized(true);
```

A.2.4 Cubo.m

```
clearall,clc

h=20;  % para abrir la figura en la ventana 20
figure(h)
hFig=figure(h)

abierta=ishandle(hFig)% Se pone a 1 si la ventana está abierta

while abierta==1;

[acc_x,acc_y,acc_z,gyr_x,gyr_y,gyr_z,mag_x,mag_y,mag_z,baud_real, roll, pitch,
yaw]=union(1);

x=[-2 2]; y=[-2 2]; z=[-2 2];

a=yaw*pi/180;
b=pitch*pi/180;
c=roll*pi/180;

% R es la matriz de Euler y E1 E2 E3 son sus transformadas

E1=[cos(b), 0, sin(b); 0, 1, 0; -sin(b), 0, cos(b)];
E2=[1, 0, 0; 0, cos(c), -sin(c); 0, sin(c), cos(c)];
E3=[cos(a), -sin(a), 0; sin(a), cos(a), 0; 0, 0, 1];

R=E3*E1*E2;


V1= [x(1);y(1);z(1)]; % Posición de R1 original
R1= R*V1;             % Posición de R1 final
```

```
V2= [x(2);y(1);z(1)];
R2= R*V2;

V3= [x(2);y(2);z(1)];
R3= R*V3;

V4= [x(1);y(2);z(1)];
R4= R*V4;

V5= [x(1);y(1);z(2)];
R5= R*V5;

V6= [x(2);y(1);z(2)];
R6= R*V6;

V7= [x(2);y(2);z(2)];
R7= R*V7;

V8= [x(1);y(2);z(2)];
R8= R*V8;

% Crea lo vectores que se representarán
X=[R1(1) R2(1) R3(1) R4(1) R1(1)];
Y=[R1(2) R2(2) R3(2) R4(2) R1(2)];
Z=[R1(3) R2(3) R3(3) R4(3) R1(3)];

X1=[R5(1) R6(1) R7(1) R8(1)];
Y1=[R5(2) R6(2) R7(2) R8(2)];
Z1=[R5(3) R6(3) R7(3) R8(3)];

X2=[R1(1) R5(1) R8(1) R4(1)];
Y2=[R1(2) R5(2) R8(2) R4(2)];
Z2=[R1(3) R5(3) R8(3) R4(3)];

X3=[R2(1) R6(1) R7(1) R3(1)];
Y3=[R2(2) R6(2) R7(2) R3(2)];
Z3=[R2(3) R6(3) R7(3) R3(3)];


h=20;  % para abrir la figura en la ventana 20
figure(h)

plot3(X,Y,Z,'b')

holdon;
```

```
plot3(X1,Y1,Z1,'r')
plot3(X2,Y2,Z2,'g')
plot3(X3,Y3,Z3,'r')

holdoff;

text(R1(1),R1(2),R1(3),'R1')
text(R2(1),R2(2),R2(3),'R2')
text(R3(1),R3(2),R3(3),'R3')
text(R4(1),R4(2),R4(3),'R4')
text(R5(1),R5(2),R5(3),'R5')
text(R6(1),R6(2),R6(3),'R6')
text(R7(1),R7(2),R7(3),'R7')
text(R8(1),R8(2),R8(3),'R8')

axis([ x(1)-2 x(2)+2 y(1)-2 y(2)+2 z(1)-2 z(2)+2])
xlabel('EJE X');
ylabel('EJE Y');
zlabel('EJE Z');
grid

pause(0.01);
abierta=ishandle(hFig);  % Indica que la figura está abierta para que se mantenga
                           ejecutando hasta que no se cierre la ventana
end
```

A.3 DIMENSIONES SENSOR MTi DE XSENS